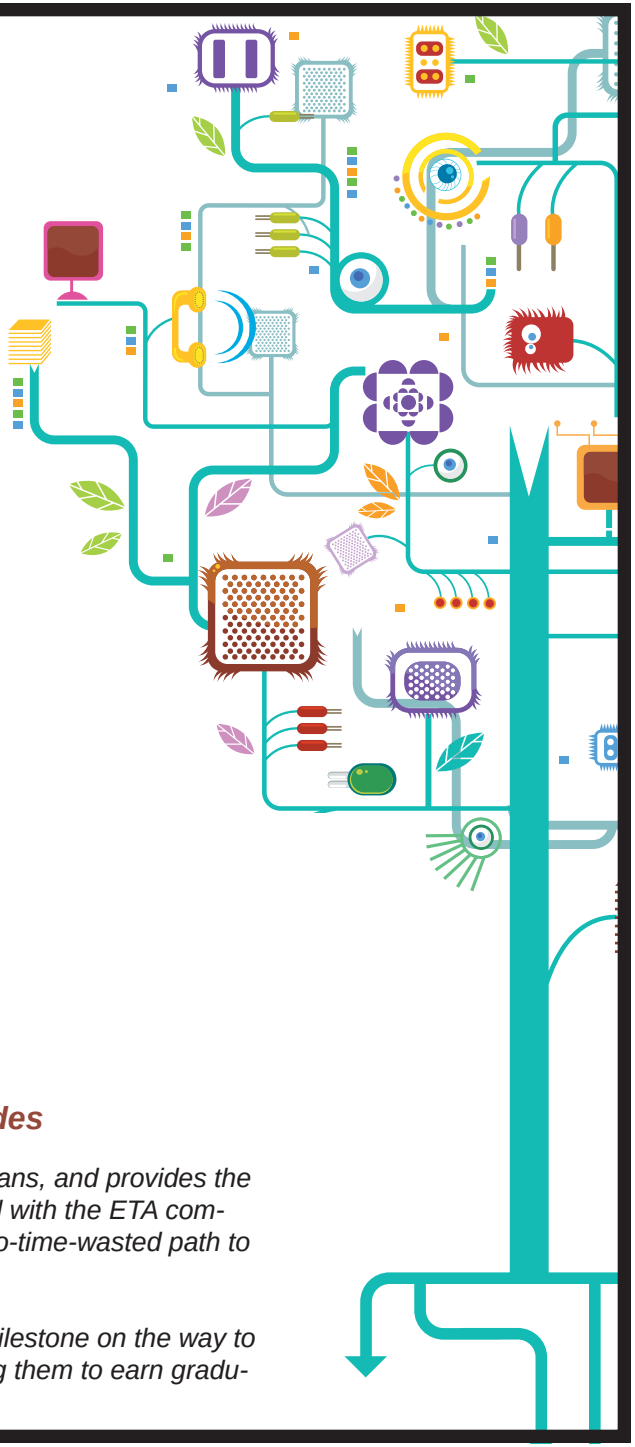
EM4 – Digital Basics

by Karl Eilers



EM4 will cover the following concepts:

- Digital concepts: analog vs. digital, decimal vs. binary
- Binary arithmetic, including octal and hex notation
- Boolean logic
- Other logic functions: bit manipulation, shifting, flip-flop etc
- Digital electronic circuitry and issues such as gate delay, race states
- Computer architecture
- Peripherals
- Programming concepts
- Microprocessors, microcontrollers, PLA's
- Machine control, servo loops, PID, PLC
- Mixed digital/analog, D-A and A-D converters
- Basic digital signal processing
- Digital generation of analog waveforms
- Servicing digital equipment

Give Your Students the Power to Succeed with the Online EM-Series Study Guides

The EM-series of study guides give students the knowledge they need to qualify as entry-level technicians, and provides the necessary foundation for further studies in specialized fields. The study guides are closely coordinated with the ETA competencies and exams. Each study guide builds on the one before it. Together they form an efficient, no-time-wasted path to knowledge and certification.

The basic goal of the study guides is the same as the goal of the EM certifications themselves: each milestone on the way to eventual mastery is useful in and of itself, giving students immediately useful information and permitting them to earn graduated certifications commensurate with their growing knowledge.

Study Guide for ETA-International EM4

DIGITAL

by Karl Eilers

Welcome to the fourth of the ETA EM6 training modules, Digital Basics. In this module, we will discuss the basic difference between analog and digital, and digital's advantages and disadvantages. You will be introduced to a different way of thinking about math, a formal system of logic, and the standard hardware building blocks of digital equipment.

EM4H.2013.1

Contents

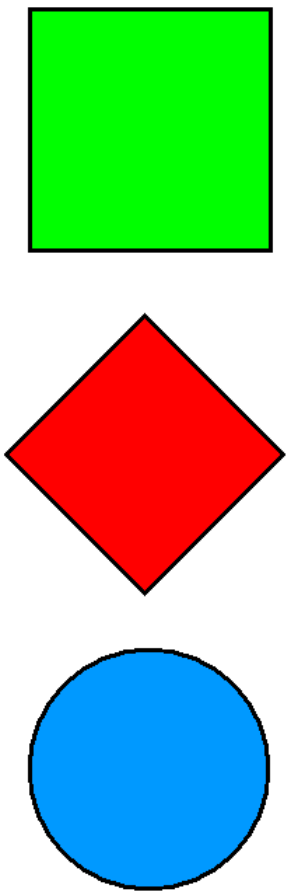
1. DIGITAL CONCEPTS
2. BINARY ARITHMETIC
3. BOOLEAN LOGIC
4. SEQUENTIAL LOGIC
5. MICROCONTROLLERS AND PROGRAMMING
6. COMPUTERS
7. MIXED DIGITAL - ANALOG
8. MACHINE CONTROL
9. DISPLAYS AND PRINTERS
10. SERVICING DIGITAL EQUIPMENT
11. REVIEW

PRACTICE EXAMS

ASPECT RATIO ADJUSTMENT

For many of the illustrations in this study guide, it is important that the dimensions and angles be correct. The green shape should appear as a perfect square, not a rectangle. The red diamond should have 90-degree corners. The blue shape should be a circle, not an oval.

If these shapes are not what they should be, please adjust your display settings before you begin this study guide.



COPYRIGHT NOTICE

All material in this study guide is protected by copyright. When you purchased this study guide, you were given an access code which grants you, and only you, the right to unlimited **personal** use of the material.

You may download a copy of this study guide at any time and duplicate it as you wish, as long as it's only for your own personal use. You may not share, post on the internet, duplicate for the purpose of sharing, use all or any part in a public presentation or in any way disseminate any part of this material to those who have not purchased the right to use it, unless you first obtain written permission from Karl Eilers or ETA-International.

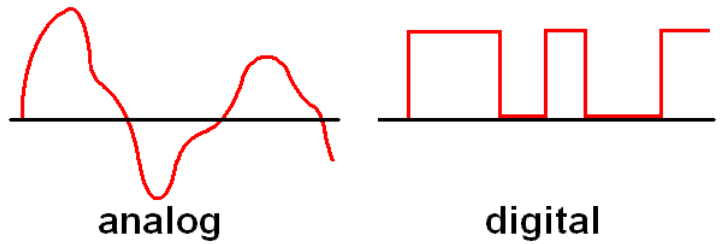
EM4 Chapter 1
DIGITAL CONCEPTS

Objectives of this chapter

- *explain advantages of digital over analog*
- *explain why most digital circuits are based on 2 and name the states*
- *define "data"*
- *define "active high" and "active low"*
- *briefly describe what the acronyms RTL, DTL, ECL, TTL and CMOS mean*
- *give proper supply voltage ranges for TTL and CMOS circuits*
- *discuss "undefined" states and give the undefined voltage ranges for TTL and CMOS*
- *discuss "shoot-through"*
- *define and discuss ESD and the precautions to be taken when servicing equipment*

To recap what was said at the beginning of the EM3 study guide about the difference between analog and digital: analog involves a continuous range of levels while digital involves specific, discrete levels. Analog is like water, digital is like ice cubes – it comes in chunks.

Digital is older than analog. We had switches before we had potentiometers; we had the telegraph before the telephone, and radio telegraph before radiotelephone (the formal name for "radio" as we know it). It takes more technical sophistication to adjust a voltage than simply switch it on or off. Digital is crude and simple; analog is refined and complex. So naturally digital came first and analog came later.



How does it happen, then, that digital has come to represent everything new and modern? The answer is, digital involves very simple circuitry, but a lot of it. An analog preamplifier requires a few transistors; a digital preamp requires hundreds. It would have been insane to amplify digitally back when you had to build circuits with discrete transistors, resistors and capacitors.

But today, an IC manufacturing plant can batch-process transistors in huge numbers. A thumb drive can contain trillions of them. It is this comparatively recent technology that has made the "digital revolution" possible. That's why digital, which is older than analog, is also newer.

DIGITAL ADVANTAGES

In many ways, digital is just better than analog. First, digital's chunkiness can be an advantage. Digital information is not degraded by small distortions or corruptions. An error has to be so great as to obscure the difference between discrete levels before it has any effect at all. This means we can transmit, manipulate and store signals with zero deterioration. And, most of the time, if a data error does occur, we can find and correct it. People often say digital information is “robust,” which is a slightly stuffy way of saying it's not delicate or easily damaged.

Second, digital and computers are natural partners. There once were analog computers, where analog values were dialed in with knobs, but computers want to be digital the way Pepsi wants to be cold. I started to write a discussion of just the most important reasons, but when it got to eight pages, I deleted it. Let's just say computers could never have gotten past the training wheels stage without going digital.

Third, we've mastered the art of *representing* analog values digitally, and converting back and forth. We can perform any analog process digitally, and this permits us to do things that would have been difficult or impossible before. For example, we routinely combine hundreds of telephone calls and send them over one communication channel, then separate them out perfectly, with no crosstalk. They used to do that with analog, but making it work was a major engineering challenge. With digital, it would take a deliberate effort to make it *not* work.

FYI, a digital information stream is not usually referred to as a “signal” but as **data**. If you want to be ridiculously correct, a single piece of digital information is a “datum.” But if you insist on being that correct, other techs will treat you like you have a contagious disease. Then again, if you like working by yourself, that might be an advantage.

DIGITAL 1 AND DIGITAL 0

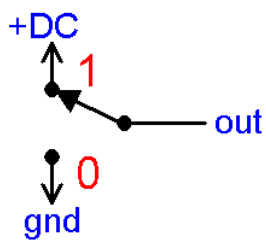
Since a digital system is based on discrete levels, we have to decide how many levels we're talking about. There is no theoretical reason we couldn't settle on five, ten, or twenty levels, twenty specific voltages between, say, zero volts and ten volts. But time and experience have shown that the most practical digital system is based on two levels: **high** and **low**. Or, if you like, on and off.

Officially, “high” is the +DC power supply voltage to a circuit and “low” is ground, and the circuit switches between them.

Having only two levels means a circuit can work like a switch. Switches are simple and cheap. In fact, two-level circuitry is so practical that when we want to represent more levels, we do it by combining several two-level circuits, as we'll see in the following chapters. As Richard Kirkland summed up in a previous ETA study guide:

Digital is on or off with nothing in between.
Analog is on or off with everything in between.

The high state is commonly represented by the number **1** and the low state is represented by the number **0**. This habit has made its way into everyday life: on my desk there's a lamp with a switch labeled "1" for on and "0" for off. But the original reason for using numbers was because the first application of binary electronics was to perform math operations.



BINARY

A system of math using just two numbers is called a **binary** system, as opposed to our familiar **decimal** system that uses ten numbers, 0 thru 9. In Chapter 2, we'll study binary numbers in depth. In the binary system the only numbers are 0 and 1. That may seem like a crippling limitation, but with just those two numbers you can do any kind of math you want. Binary has so many advantages that, nowadays, "digital" almost always means "binary."

ICs AND DIGITAL LOGIC

The catch-all name for digital computing is **logic**, a word which applies to the circuitry itself and also to the operations that it performs.

When we talk about digital circuitry, we're almost always talking about integrated circuits. Before ICs, digital was done with discrete transistors. In fact, the earliest electronic computers used tubes. The first computer that was not just a laboratory curiosity, but was built for actual use, operating with changeable programs, was the ENIAC from 1946. It had over 10,000 tubes and occupied a warehouse. One of the industry gurus at that time made a prediction that there would be a worldwide market for something like half a dozen computers. With the technological limitations of the time it was a reasonable prediction. Nobody foresaw that ICs would reduce the volume of computer circuitry by – just a guess – a factor of 10^{16}.

ICs have put computers in everybody's hands. More important, perhaps, they have made possible many digital things that are not computers, like cellphones and the worldwide infrastructure that supports them.

PULL UP AND PULL DOWN

As mentioned, the output of a digital circuit at any given moment will be connected either to the +DC supply or ground – or as engineers say, it will **pull up** or **pull down** at the output.

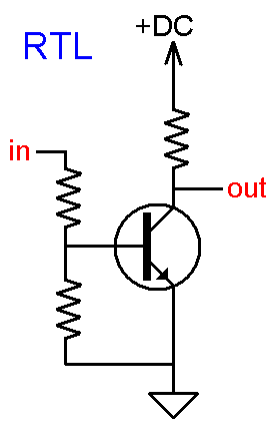
There's a limit to how hard an output can pull up and down. For example, a standard computing circuit cannot pull hard enough – it cannot create enough current flow – to drive a relay. Some kinds of output can pull harder one way than the other, and some kinds of input require more pull than other kinds. This ought to be an engineer's problem, but not all engineers are brilliant, meaning a tech sometimes has to ferret out problems that were built in at the design stage.

LOGIC FAMILIES

Digital integrated circuits come in several families, which are grouped by the particular kinds of transistors and other parts they contain. I'm going to go back into the past a ways because some of today's practices are leftovers from earlier designs, and because you may run into some very old machinery in the field. Just a few years ago I had to repair a machine that had been operating unchanged since the early 1970s.

RTL

The first digital ICs came out back when hippies smoked pot and Shelby Cobras smoked everybody. They were originally made with resistors and transistors, just standard discrete circuits converted to integrated form. **RTL** stands for **resistor-transistor logic**, that is, ICs built with transistors and resistors.



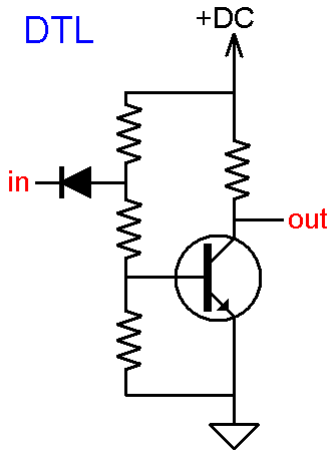
At the risk of getting way ahead of myself, I'm going to show you the simplest RTL circuit. If you've gone through the EM3 study guide, you should be able to look at voltages and figure out how it operates. This is an **inverter**, a circuit that turns a 0 into a 1 or a 1 into a 0. You'll meet this function again in Chapter 3. In an RTL output, a transistor pulls down to make an 0 and and a resistor pulls up to make a 1, meaning that it can pull down harder than it pulls up – you could say it has a stronger 0 than 1.

On the other hand, the input circuit is a voltage divider that requires a pullup. The driving circuit doesn't have to pull down on the input at all to get a 0, but has to pull up to get a 1. The combination of pull-down inputs and weak-pull-up outputs means a digital “1” isn't always reliable, especially when an output must drive more than one input. (The number of inputs that an output can drive is called its **fanout** capability. An output that can drive, say, four paralleled inputs of the same type of logic family has a fanout of 4.)

The inverter is an example of something that's also true of all the other, more complicated digital circuits: it's based on a transistor acting as a switch. If you've gone thru the EM3 study guide, you know a transistor is a valve which can be “opened” to any desired degree to let current through. But it can also be completely closed, or it can be opened all the way, so the entire supply voltage appears across the collector resistor. This is called “saturation,” which is defined as the condition where collector voltage is almost the same as emitter voltage. Transistors, acting as switches, are the basic building blocks of digital circuits.

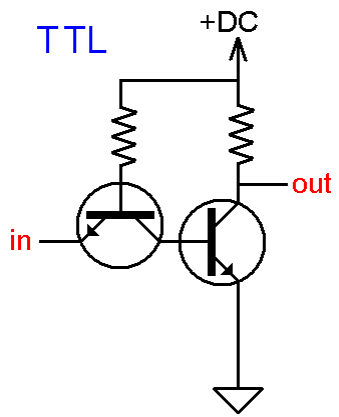
RTL is completely obsolete; it is only of interest today to show the progression of IC technology.

DTL



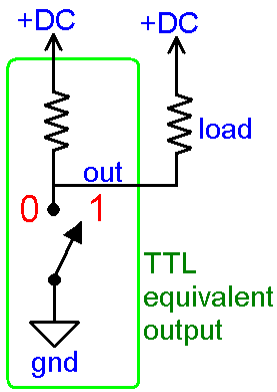
The next step was DTL, which stands for **diode-transistor logic**. In DTL, the input is constantly pulled up by internal resistors, so it normally sits near “1.” To input an 0, the driving circuitry pulls down on the input through a diode. This creates a better match between inputs and outputs, so multiple DTL circuits work together better than multiple RTL circuits. It also increases speed. DTL is also obsolete today.

TTL



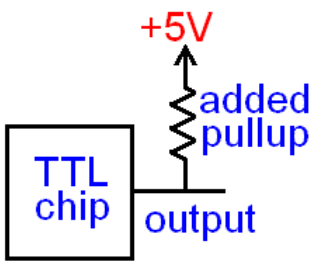
TTL, or **transistor-transistor logic**, is the earliest IC family that techs will still find themselves dealing with. The reason for that strange name, “transistor-transistor logic,” is to distinguish it from RTL and DTL chips that were still in use when TTL was introduced.

In TTL chips, transistors take the place of most resistors and diodes. The main advantage is faster operation than RTL and DTL. Later versions used Schottky transistors to increase speed, and higher internal resistances to reduce power consumption.



In all of the families described to this point, every output is a pull-down output – an NPN transistor turns on and pulls the output to ground. When it's in the "1" state, a resistor or another transistor pulls up, but the pull-up is so weak that, if the output has to supply any significant current, an external pull-up load resistor may have to be added.

If you're ever working on an old TTL circuit that's glitchy and refuses to behave, first check to see that the DC supply is well filtered and regulated. Then try clipping a 2.2k resistor to a +DC supply rail and touching the other end to an output at the point in the circuit where things are going wrong.



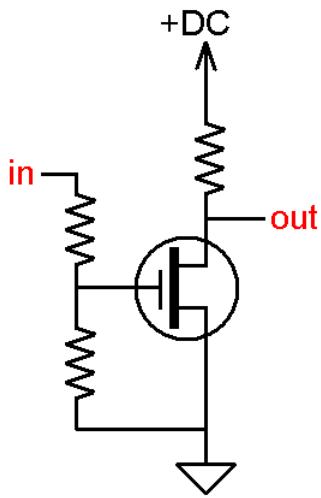
The TTL input, like its output, is pull-down: it is a transistor emitter that is normally in the digital 1 state and must be pulled down for a digital 0.

ECL

One other bipolar transistor family merits a brief mention. **Emitter-coupled logic** is a very high-speed bipolar technology. It also draws a lot of current and costs a lot of money, so it's only used where the fastest possible operation is required. Now that we can get CMOS to work at several GHz, ECL is not much used any more.

MOS

If the BJT transistors are replaced with FETs, we can have digital circuitry that draws a lot less current. Junction FETs would require a bipolar supply, so we'll be using enhancement-mode MOSFETs. The circuit to the right is a MOSFET inverter.

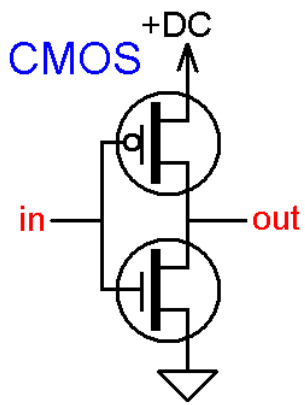


MOSFET circuitry like this is widely used to control machines, as we'll see later. But for logic, we want to go one step farther. Whenever the output is low, the MOSFET draws current through the resistor.

If we can provide a pull-up component that only pulls up when the output is supposed to be high, and turns off when the output is supposed to be low, we'll cut the current consumed by the logic to almost zero. That component is a FET of the opposite polarity, a P-channel MOSFET.

CMOS

CMOS means **complementary metal oxide semiconductor**.
"Complementary" means that the circuit combines N-channel and P-channel MOSFETs. In the illustration, the upper transistor symbol, with the little circle, is an alternate way of drawing a P-channel MOSFET that is often used in digital circuit diagrams.



CMOS is today's technology, universally used by computers and digital circuitry. It's not hard to see why.

- CMOS inputs are essentially open circuits, so they don't significantly load whatever is driving them.
- CMOS outputs, being complementary, pull both up and down with equal strength, like an SPDT switch.
- When a CMOS circuit isn't switching – when it's sitting at 1 or 0 – it draws essentially no current.

Modern computing wouldn't be possible without CMOS. BJT (bipolar junction transistor) technologies like TTL draw power at all times. Even if we could put billions of BJTs in a tiny space, that much concentrated power would melt everything. CMOS, in contrast, draws so little current when it's idle that the voltage necessary to store data for weeks can be supplied by a capacitor.

VOLTAGES

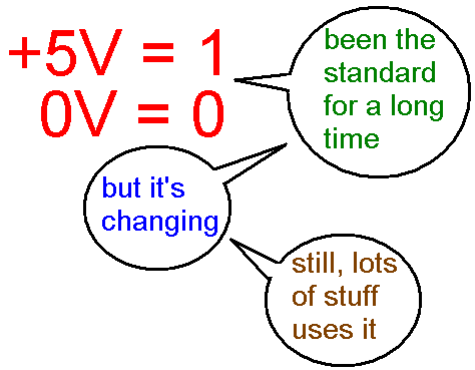
Binary 1 and 0 aren't just concepts, they're actual high and low voltages. For most digital circuitry, the operating power source is a single positive DC voltage. This means there are two power supply rails, +DC and ground.

The circuitry switches back and forth between these two rails; digital 1 is the +DC voltage, no matter what that happens to be, and digital 0 is ground, that is, zero volts.

(Note – ECL is the exception; it is the only digital family that runs on a negative DC supply voltage.)

Long ago, +5V was chosen as the "standard" DC voltage because that's what TTL runs on. To misquote Henry Ford, you can power TTL with any voltage you want, as long as you want 5 volts.

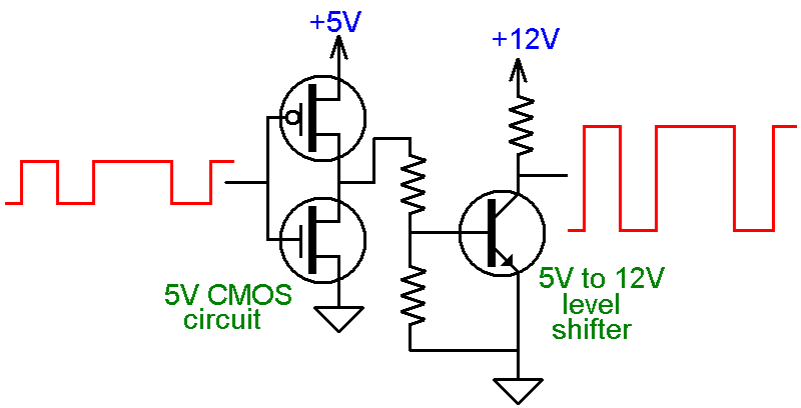
That limitation does not apply to CMOS. Standard CMOS will run on any voltage from about 3V to about 18V, though the higher voltages pose a danger I'll talk about later. However, 5V has been in use for so long that it's firmly established in people's minds as the standard DC voltage for CMOS as well as TTL.



In today's world, computers actually run on 3V or less. If you lower the DC voltage on standard CMOS, it gets slower. But if you design CMOS for a low supply voltage in the first place, it can be very fast. It takes time for a circuit to change voltage. This is called the **slew rate** and it's measured in volts of change per microsecond. If the DC voltage is reduced to 3V, a circuit with a given slew rate can cross the distance between 3V and 0V in less time. Also, lower voltage means lower power dissipation, less heat, smaller size (because the less heat generated by the circuitry, the more circuitry you can safely stuff into a given space), and longer battery life.

Still, a lot of equipment uses +5V. In this study guide, I'm going to treat 5V as the standard voltage, and trust you to remember that you'll encounter other supply voltages in the equipment you work on.

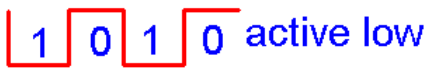
Whatever the DC supply voltage happens to be, that's digital "1" and 0V is digital "0." If the supply voltage is different from the supply voltage that runs the rest of a piece of equipment, the two can be interfaced by means of a **level shifter**.



LEVEL SHIFTER

ACTIVE HIGH, ACTIVE LOW

Got all that? Okay, now I'm going to throw in one more factor. Digital "1" is usually thought of as "**on**" or **active**. Up to this point, I've been talking about **active-high** circuitry, where "1" is the higher of two voltages. There is also such a thing as **active-low**, where 1 is the lower voltage, i.e. ground. I'm not going to go into the reasons why a designer would choose to work upside down, but you should be aware that such a thing exists. In this study guide, I'm mostly going to stick to active-high circuitry, but as you'll see in Chapter 3, even with active-high circuitry, an individual input or output may be active-low.

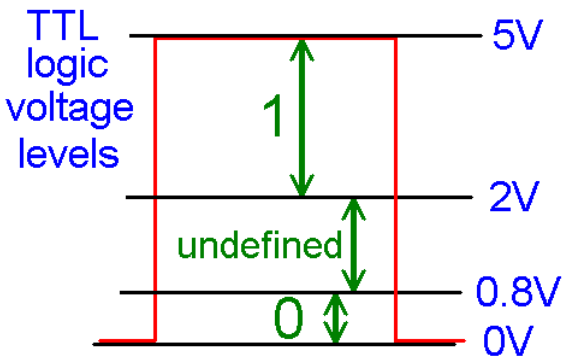


1, 0, AND TROUBLE

Digital output circuits, especially when loaded, cannot pull all the way up to 5V or down to 0V. How far can we depart from those voltages before the circuit gets confused?

Undefined TTL Voltages

With TTL, a floating, open input will sit near +5V or digital 1 and must be pulled down for 0. The rule is, anything below 0.6 to 0.8V is a reliable "0" and anything above 2V is a reliable "1."



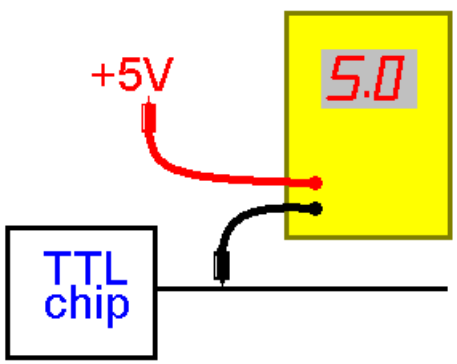
The range between 0.8V and 2V is a no-man's-land labeled **undefined.** If we start with an input at ground and gradually increase the voltage, somewhere around 1.4V the output will switch – but we don't know exactly where. While the input is in the undefined region, the output will be unpredictable and erratic.

Equipment is supposed to be designed so voltages switch between 0 and 1 quickly, spending as little time as possible in the undefined area. However, with poorly designed equipment, this is not always the case. In particular, TTL's weak pullup means that any noise source in the vicinity can put a negative-going spike on an input that goes far enough into the undefined region to cause a glitch at the IC's output.

I mentioned the old machine I had to fix. What I did was, I went through the circuit and put a 1k external pullup resistor on every output where the problem might be occurring, to provide a strong enough “1” that the circuit would never drop into the undefined zone. Even with the extra pullup, the TTL outputs still were strong enough to pull down, with current to spare.

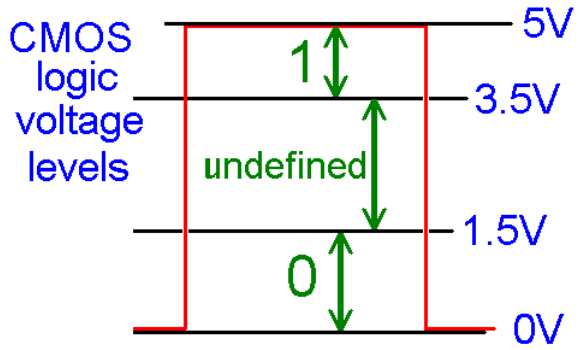
(BTW – the machine was making wires for 1970s-era control systems that are currently being installed in *new* aircraft. Something warm and fuzzy to think about next time you fly.)

If you ever find yourself checking logic levels in a TTL circuit with an analog voltmeter (you should be using a logic probe, but sometimes there isn't one available), bear in mind that, due to the weak pullup of the inputs, the meter resistance will often turn a “1” into a “0,” or at any rate pull it down into the undefined region. In this case, better to measure the voltage by connecting the meter between the input and the + supply, then subtracting to find the difference.



Undefined CMOS Voltages

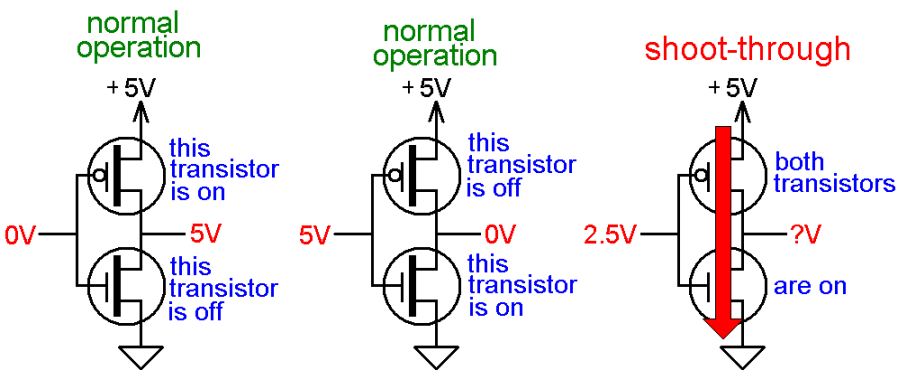
With standard CMOS, “0” is any voltage within 1.5V of ground and “1” is any voltage within 1.5V of the +DC rail;if the rail is 5V, anything over 3.5V.



The undefined region, then, is between 1.5V and 3.5V. I keep saying “standard” CMOS because there are variations designed to do special things, and one of those has logic levels that correspond to TTL, so the two can be used together in a circuit. And nowadays, with lots of CMOS designed to work on 3V or lower, the limits of the “defined” voltages are closer to the rails. Sometimes the subtle differences among CMOS types cause a chip that should work in a circuit not to work, in which case you'll need to look closely at the part number to see which CMOS version you're replacing. This may someday save you from tearing your hair out when you're repairing a piece of equipment.

Shoot-through

With CMOS, an input in the undefined zone poses an additional danger. CMOS devices are enhancement-mode FETs. If the input voltage is 1.5V or less, half the FETs in the chip will be off. If it's 3.5V or more, the other half will be off. But between 1.5V and 3.5V, all transistors are on. That means they're creating a direct current path across the power supply. This is called **shoot-through** and with CMOS it happens during every transition. Shoot-through current is greater than what the chip can handle on a continuing basis. If it lasts for any length of time, the chip melts.



If the supply voltage is higher than 5V, the danger posed by shoot-through gets a whole lot worse, because the FETs turn on harder – pass more current – and drop more voltage. The reason they don't normally melt is because, in a well designed circuit, inputs spend microseconds or less in the undefined region.

The number one rule when using CMOS is that an input must never be left floating; there must be a DC path that pulls it to one rail or the other.

Here's something to bear in mind when servicing a CMOS circuit: inexperienced techs often destroy a CMOS chip in the course of probing it with an oscilloscope or meter. That's usually blamed on static discharge, but it's often caused instead by accidentally pulling an input into the undefined zone.

With a 12V or higher supply, the resulting shoot-through will destroy an IC in milliseconds. You should never attach a test lead to a CMOS input until you've looked at the schematic and verified that the voltage divider formed by the unit under test and your test equipment won't pull the voltage into the undefined zone. If you're using an oscilloscope, put the probe switch in the 10X position. That corresponds to a 10 megohm input resistance, which almost never causes problems with CMOS.

The number two rule is that rise and fall times should be as close to instantaneous as possible. That will minimize the duration of shoot-through. Also, some circuits will not operate unless the transitions are fast.

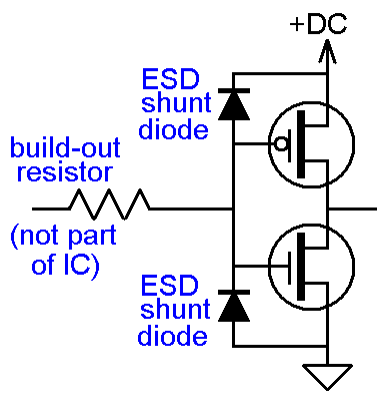
STATIC DISCHARGE

Then again, the damage may actually be caused by static. When working with CMOS, be careful about static. FET gates are open circuits, and they're very vulnerable.

The formal name for what happens when static jumps is **electrostatic discharge**, or **ESD**. If a CMOS chip were a field mouse, ESD would be a snake lurking in the grass, waiting to strike.

Most CMOS chips now contain ESD diodes, which are there to drain off ESD spikes to the DC or ground rails. But don't depend on them. Wear a wrist strap with a 1-meg resistor in it, wired to Earth ground. Ground soldering irons and other tools. If you're going to work with this stuff enough to make it worth the bother, get a metal-topped bench that's Earth grounded through a 1 meg resistor.

A well-designed circuit will often have a discrete resistor between a CMOS input or output and the rest of the circuit. This is called a **build-out resistor** and it's there to provide extra protection against ESD.



NUMBERING SYSTEMS

TTL ICs have a standard part numbering system which consists of two digits, usually either 54 or 74, followed by two or more numbers that identify the particular function. For example, a 7404 is always a chip with six inverting circuits inside it like the one I showed you on page 4.

The generic number is preceded by letters which identify the manufacturer, as in CD7414 or SN7414. As time goes on, these identifiers become increasingly general as new manufacturers enter the market and old ones exit.

Improved versions are identified by additional letters inserted after the 54 or 74. For example, early TTL technology was replaced by **low-power Schottky** TTL, which would be identified by a number like SN74LS14. When you run into something unfamiliar, you can generally find information on the web.

By the way, whoever chose “74” as the initial numbers was an idiot. The date of an IC's manufacture is a four-digit code stamped on the IC that names the year and week; 9412 means the 12th week of 1994. TTL ICs manufactured in 1974 have two numbers that begin with 74, leading to all sorts of confusion.

Many a tech has tried to cross-reference a 74- number that doesn't exist. Something to be aware of if you work on equipment that old, which fortunately doesn't happen much any more.

CMOS ICs were originally identified by numbers beginning with 40, e.g. the 4066, a quad analog switch. Many CMOS chips are also identified by 74- numbers because they are CMOS versions of standard TTL chips, and have the same connection pattern (called the **pinout)** as their TTL equivalents. Such an IC will have a 74- number with a "C" in it, as in 74C14. More advanced versions have more letters, such as 74HC14 or 74AC14.

A CMOS 74XX chip with a "T" in the number indicates that it's designed to interface well with TTL. Most of these will not work well with supply voltages higher than 5V. Letters after the part number identify the case style and other parameters. "CN" is a common identifier, meaning that the IC has a plastic DIP case and a normal operating temperature range (as opposed to the more demanding military temperature range).

There are many variations which, depending on the particular piece of equipment, may or may not be interchangeable. You can find information about a particular chip on the internet, and the schematic diagram of the equipment you're working on will generally tell you whether a particular version is required.

Some ICs are **house numbered**, meaning that their numbers are nonstandard and probably not published. They may actually be custom chips, or they may be standard chips with nonstandard numbers to "encourage" you to buy replacements from the equipment manufacturer. This is not necessarily a scam; sometimes, especially in life-safety equipment, house-numbered chips are standard chips that have been subjected to especially rigorous testing procedures.

You will probably learn to look at diagrams and figure out what these standard chips are, but with life-safety equipment, it's best to replace a defective chip with an identical house-numbered part from the equipment manufacturer. If nothing else, this keeps you from exposing yourself to one particular kind of legal liability.

WHAT'S NEXT

In chapter 2, I'm going to show you how computers count in 1s and 0s. In chapter 3, I'll show you how they think – or, at any rate, how they embody the thoughts of their creators.

End of chapter 1

EM4 Chapter 2
BINARY ARITHMETIC

Objectives of this chapter:

- *discuss the differences between binary and decimal notation systems*
- *explain "base," "radix," "place holder" and "point"*
- *define "bit," "byte," "nibble," "word"*
- *define LSB and MSB*
- *give the 12 addition, subtraction, multiplication and division rules and demonstrate their use*
- *discuss and compare decimal, binary, octal, hexadecimal numbering systems, and demonstrate conversions*
- *define and discuss EBCDIC, ASCII and UTF-8*

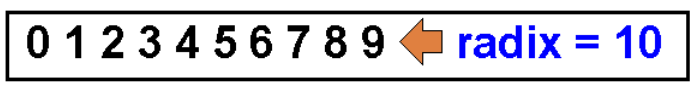
As shown in chapter one, computers usually work with two voltage levels, which we call **1** and **0**. We could have called them + and G for positive and ground, or H and L for high and low. But using numbers to represent voltages makes it easy and natural for computers to do number problems.

However, we have to use a number system that corresponds to the voltages actually found in a computer, and there are only two: +DC and ground. Thus the need to use a number system based on only two numbers. Our normal numbering system is called **decimal** because it's based on the numbers 0 thru 9, ten in all, thus "dec" as in "decade," ten years. The word for a system based on two numbers is **binary** - "bi" meaning two, as in the two-wheeled vehicle, the bicycle.

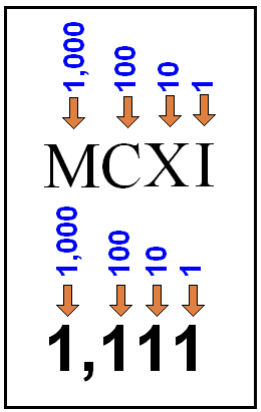
It is not immediately obvious that binary is the best system for calculation. Counting in tens seems so natural to ten-fingered humans that it's hard to imagine counting any other way. ENIAC, the first seriously useful digital computer, used decimal math, probably not because its designers were blinded by habit but because it was intended for real-world use, and in 1946 there were few users accustomed to working in binary.

As well suited as decimal numbers are for people, they're not so hot for computers. An electronic circuit that "thinks" in ten voltage levels is complicated and clumsy. A circuit that thinks in two levels – on and off, high and low, +DC and ground, 1 and 0 – is simple, cheap and fast.

To get a good handle on binary, we first need to review a few things about the decimal system. As it is based on ten numbers, it's known as a **base 10** system. Mathematicians have another word for base, which is **radix**. It's in the exam.

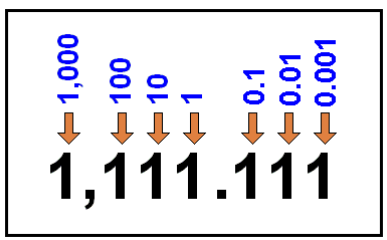


Our numbering system is a **positional** system: in a large number with many digits, the significance of a digit depends on its place within the number.



By comparison, Roman numerals are not positional. One is I, ten is X, one hundred is C, one thousand is M. But in our decimal system, one, ten, one hundred and one thousand are all denoted by the number "1," and we tell what a "1" means by where it sits, i.e. how many digits there are to the right of it.

"1" can also mean a tenth, a hundredth, or a thousandth, if it's positioned to the right of the ones place. These are **decimal fractions**, and the dividing line between **whole numbers** (also called **integers**) and fractions is indicated by a dot called the **decimal point**.



Thus the three most important characteristics of a positional number system like ours are:

- The **base** or **radix** – the range of numbers the system is based on. In our decimal system, they are the ten numbers zero thru nine, so it's a **base-10** system.
- The **place holder** – the principle that the significance of a number depends on how many places are to the right of it. If there is no quantity in one of those places, a zero is inserted to "hold" the place. The number 1,000 has a 1 in the fourth place, meaning the 1 signifies a thousand, and the three zeroes are place holders. In a decimal system, a system based on ten numbers, each place is **ten times more significant** than the place to the right of it.
- The **decimal point** – the dot that indicates the division between positions representing whole numbers and positions representing fractions, permitting numbers smaller than one to be notated as decimal fractions by writing numbers to the right of the point. Each place to the right of the point is **1/10th the significance of the place to the left of it**.

The left-most digit is the **most significant digit** or **MSD** and the right-most digit is the **least significant digit** or **LSD**. Or, if you're a child of the hippie era like me, LSD could mean... oh, never mind.

The binary system works exactly the same way: it's a positional system, the significance of a digit depends on how many digits there are to the right of it, zeroes are used as place holders; there's a point, and fractions are written as numbers to the right of the point.

The difference is that, instead of using ten numbers, the system is based on only two numbers, zero and one. This changes a few things:

- The binary system is a **base-2** system; the radix is 2.
- Each place is **twice** the significance of the place to the right of it.
- Each place to the right of the point is **half** the significance of the place to the left of it. Of course, we just call the point a point, not a decimal point, because this isn't decimal.

Let's count. In binary, counting customarily starts not with one but with zero. After all, 00000000 is just as legitimate a set of digits as 00000001. Later, when we study binary data that represent things other than numbers (computer commands, for example) it will become obvious that there's no good reason to treat a number that's all zeroes differently from any other. Computers start counting at zero, so we need to adopt that mindset.

In the "ones" place we can have either a 0 or a 1, so our first two counts are 0 and 1. If we want to count higher, we create a new place to the left. A 1 in that place represents two. We will need a placeholder in the ones place, just as we do in the decimal system, so the decimal number 2 would be 10 in binary. We don't call this “ten,” because it isn't ten, it's two. We call it “one oh” or “one zero.”

counting in binary
0 zero
1 one
10 two
11 three
100 four
101 five
110 six
111 seven

Suppose we add a 1 in the ones place, turning the zero into a one. Two plus one is three, so binary 11 represents three.

If we want to represent four, we have to add another place. Binary 100 – a one in the "fours" place, with two place holders, represents four. 101 is five, 110 (four plus two) is six, 111 (four plus two plus one) is seven. Then we have to add an "eights" place to go further.

Thus:

- In the decimal system, going from right to left, least significant to most significant, the places represent multiples of ten: one, ten, one hundred, one thousand and so on.
- In the binary system, they represent multiples of two: one, two, four, eight and so on.

In the binary system, we don't insert commas in big numbers. In the decimal system we are used to thinking in groups of three orders of magnitude – thousands, millions, billions – but this concept doesn't apply to binary.

1010010 is just "one oh one oh oh one oh." So there's no reason to put in a comma every three digits.

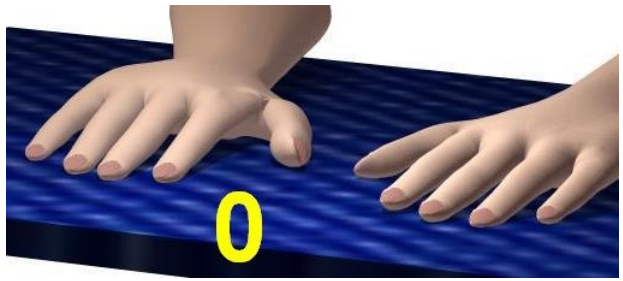
Let's see if you've got it so far: what does the digital number "1010010" represent?

There's nothing in the ones place, but a 1 in the twos place, so that's two. There's nothing in the fours or eights place, but a 1 in the sixteens place. Sixteen plus the two we already had make eighteen. There's nothing in the 32's place but a 1 in the 64's place. 64 plus the 18 we already had is 82. That is, binary 1010010 is decimal 82.

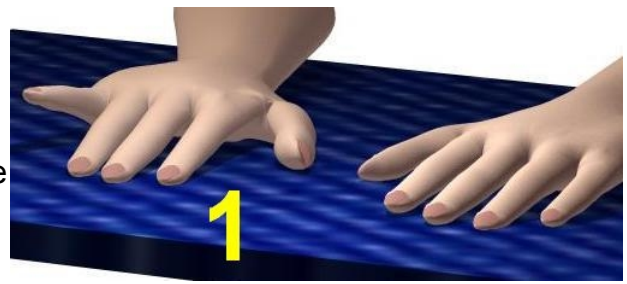
1010010
64 16 2
64 + 16 + 2 = 82

Binary seems like an inefficient way to write numbers. It takes seven digits to represent a binary number that can be represented with two digits in decimal. However, a computer designer would say that seven simple circuits are easier and cheaper and work faster than two complicated circuits.

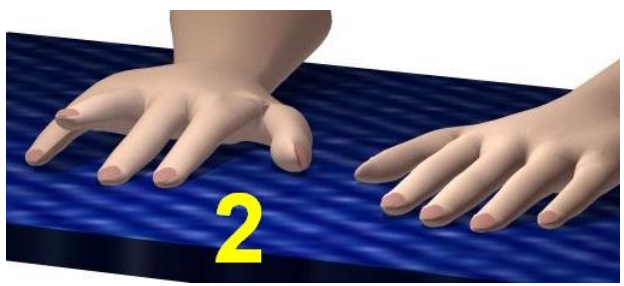
Here's an exercise to prove how efficient binary really is. In decimal, you can count up to ten on your fingers, right? Okay, now let's count in digital. Put your hands down on the table in front of you. Your ten fingers represent the binary word 0000000000.



Now lift your right pinky. You have incremented the count to 0000000001.



Put your pinky down and raise your fourth finger if you can. (Most people have trouble doing that.) Your fingers now represent 0000000010, or decimal two.



Now, keeping the fourth finger up, raise the pinky again. Your fingers now represent 0000000011, or three.

If you were to keep this up, counting in binary until all fingers are lifted, you would eventually wind up with the binary number 1111111111, which is the decimal number 1,023. So: in decimal, you can count to ten on your fingers. In binary, you can count to 1,023. It wouldn't be stretching the point too far to say that, far from being inefficient, binary is 100 times more efficient than decimal because *it makes efficient use of counting equipment.*

That's important, because there is a fundamental difference between the way people think and the way computers think. People use the same brain for all thoughts, and one thought isn't more expensive than another. You use the same brain to write a grocery list that you use to write a novel.

But in a computer, every bit of “thinking” power has to be manufactured, which means it represents a physical circuit that has a dollar cost. Computer memories (which you could call “remembering” power) have trillions of circuits, so if we can save a thousandth of a cent on each circuit, that's a thousandth of a cent saved trillions of times over. There's no way we could build today's electronic gear if we weren't being as efficient as possible.

Digital functions involve simple steps but lots of them. Think of coral, which are primitive, tiny sea creatures that do nothing but stay in one place all their lives, filter nutrients from the water, and die, leaving behind their tiny skeletons. Gazillions of those skeletons are what form coral reefs, which are larger than any structure ever built by humans – a demonstration of the power of micro-events in mega-quantities. Similarly, binary computing means taking very small, very simple steps, but taking a lot of them, very quickly. Computers are stupid but fast. The things we do with digital technology are the result of huge quantities of simple operations performed at a rate of a billion or more per second.

In the following chapters, there will be times when we'll go back and forth between decimal and binary, and also talking about binary-related numbering systems called **octal**, a base-8 system, and **hexadecimal**, a base-16 system. In these cases, where there is a possibility of confusion, I'm going to indicate a number's base or radix by a subscript: $1{,}000_{10}$, 100101_{2}, 23_{8}, $4F_{16}$. (Yes, 4F is a number. You'll see.)

BINARY ARITHMETIC

The advantage of binary numbers becomes obvious when we start talking about adding, subtracting and so on. Binary arithmetic is so mechanical you can almost see the gears spinning. It's the perfect system for a calculating machine.

You don't need to study the following material very hard, a tech isn't likely to actually have to do binary arithmetic very often. You tell a computer chip to add and it adds. But it's worth the time to develop a feel for how binary arithmetic is done.

ARITHMETIC TABLES

I don't know if kids have to memorize addition and multiplication tables any more, but I did. It's not necessary to memorize sums and products involving zero or one, or duplicate combinations (3x7 is the same as 7x3). That still leaves dozens of addition sums and dozens of multiplication products to learn, and then we have to figure out subtraction and division.

With binary, the addition and multiplication tables are simple. Note that the ones with the red pointers are the same in binary as in decimal.

For addition, there are only three possibilities:

0 + 0 = 0 ◄
1 + 0 or 0 + 1 = 1 ◄
1 + 1 = 0, carry 1

For multiplication, only three possibilities:

0 x 0 = 0 ◄
1 x 0 or 0 x 1 = 0 ◄
1 x 1 = 1 ◄

For subtraction, there are four:

0 – 0 = 0 ◄
0 – 1 = 1, borrow 1
1 – 0 = 1 ◄
1 – 1 = 0 ◄

With division, two of the possibilities are illegal, because they involve dividing by zero.

0 / 0 = forbidden
1 / 0 = forbidden

That leaves two:

$$0 / 1 = 0$$
$$1 / 1 = 1$$

So, the entire single-digit binary addition, multiplication, subtraction and division table has 12 values. With only two levels, 1 and 0, and with only 12 possible outcomes, it's easy to build circuits to perform every possible calculation.

BITS AND WORDS

Digits come in batches. In the decimal system, a single digit can represent ten values, two digits can represent 100 values, three digits can represent 1,000 values and so on. Binary works the same way, but with only two possible values per digit, a single digit can represent only two values, 0 and 1. Two digits can represent four values: 00, 01, 10 and 11 (zero, one, two, three). Three digits can represent eight values, four digits can represent sixteen values and so on.

Each binary digit is a **bit**. A group of bits is a **word**.

Word size matters because, as noted above, every bit requires circuitry. A computer can't think about an 8-bit number like 1001011 if it only has six thinking circuits. So, when a computer is designed, the designers have to decide what word lengths it will accommodate. Short words mean a computer that's limited, long words mean a computer that's expensive. Once a standard word length is decided on, all the thinking circuits are built to that standard.

Because computers so naturally work with increments of two, word lengths tend to be multiples of two bits long: 2, 4, 8, 16, 32 bits etc. The larger the standard word – the more binary digits – the larger the numbers that can be represented, just like decimal. An 8-bit word can represent any decimal number up to 255_{10}; a 16-bit word up to $65{,}535_{10}$; a 32-bit word any number up to a bit over 4 billion$_{10}$. In binary, the left-most place is the **most significant bit** or **MSB** and the right-hand place is the **least significant bit** or **LSB**.

This is similar to what happens with decimal numbers. The difference is, with decimal each added place increases the numbers that can be represented by an order of magnitude. With binary each added place doubles the number that can be represented, as shown in the chart on the next page. Note that, in binary, the largest value that can be represented is always one less than a power of two. But the total number of possible values, counting zero, is always a power of two.

no. of bits	*no. of possible values*	*lowest binary value*	*highest binary value*	*highest decimal value*
1	2	0	1	1
2	4	00	11	3
3	8	000	111	7
4	16	0000	1111	15
5	32	00000	11111	31
6	64	000000	111111	63
7	128	0000000	1111111	127
8	256	00000000	11111111	255

In the 1960s, the 8-bit word became the standard chunk of data, known as a **byte**. Even today, in an era of 16, 32 and 64-bit words, a lot of computing is done in chunks of 8 bits. 4 bits is a small "bite" of data known as a **nibble** or nybble. You'd think a nibble, which can represent only 16 values, would be too small to bother with, but in fact we deal with nibbles all the time.

Computer geeks, thinking they were being cute, gave a single binary digit and a chunk of eight digits names that both begin with "b," which means shortening "bit" and "byte" to one letter abbreviations leads to confusion. The rule is, a lower-case **b** means a bit; an upper-case **B** means a byte. Do people reliably follow this rule? Ha ha, good joke. Sometimes you will have to do a little investigation to figure out what that "b" or "B" actually means.

BINARY-BASED NOTATION SYSTEMS

Computers work very happily with two levels, but it is natural for humans to want to represent numbers by the familiar decimal digits 0-9, so we can work with them. Several notation systems have been designed to permit this.

OCTAL NOTATION

Our decimal numbers 0-7 can represent up to a three-bit binary word:

000 = 0
001 = 1
010 = 2
011 = 3
100 = 4
101 = 5
110 = 6
111 = 7

It would seem that by adding a bit we could keep going, since $1000_2 = 8_{10}$, but we can only go as far as $1001_2 = 9_{10}$ before we run out of decimal numbers. If we want to keep everything in multiples of two for the sake of efficient computing, we should stop at 111_2 and let a single decimal digit represent only numbers from 0 to 7_{10}.

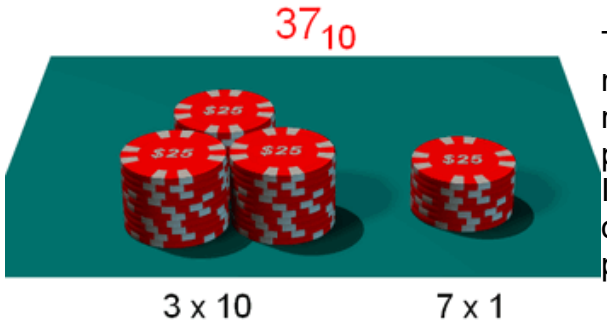
Larger values bring a second 3-bit number into play. We call these **octal** numbers, because there are 8 possible values, so this is a base-8 system.

An octal digit can represent any binary number up to 111_2, which is to say any 3-bit binary number. 111_2 is 7_{10}. The next octal digit represents $8_{10;}$ that is, the second place is the "eights" place. Two octal digits together can represent 8 x 8, or 64_{10} different numbers, 0 thru 63_{10}, which would be $111\ 111_8$.The third digit would be the "sixty-fours" place.

For example, consider the digits in octal number $74_{8.}$ 7_8 converts to binary 111_2 and 4_8 converts to 100_2, so the total binary number is $111_2\ 100_2$ or 111100_2.

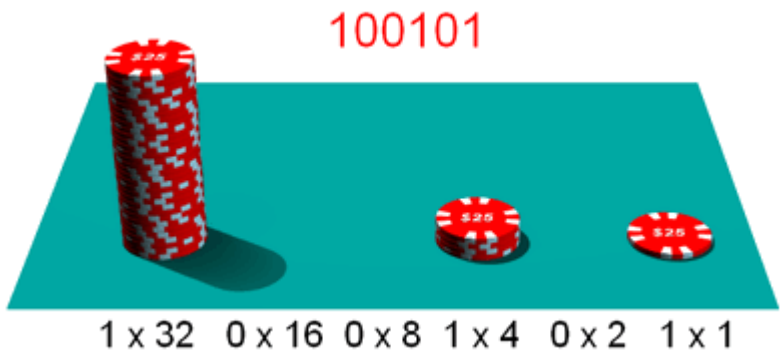
I have to emphasize that in this context, "74" does not mean seventy-four. It means octal seven four, binary 111100, decimal 60. Octal 74 is not a decimal number. It's a way of using number symbols we're already familiar with to represent a binary number.

One of the keys to understanding this conversion business is to realize that "1" is always a single unit of something, like a poker chip. When we change numbering systems, we aren't changing the total number of chips, we're just stacking them differently.



Take the decimal number 37_{10}, which means three "tens" plus seven "ones." Imagine three stacks of ten poker chips, plus seven more.

Let's convert to binary. In the decimal system, we had three stacks in the 10s place, but we can't do that in binary; there's no such thing as a "three" - only zero and one. So

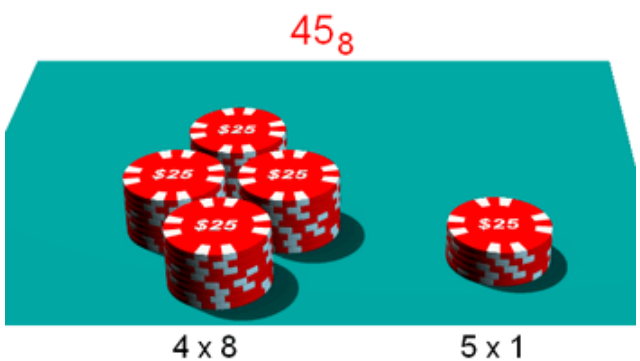


there can be only one stack of chips in any given place. In a binary word, the sixth bit from the right is the 32s place. Let's make a stack of 32 chips. If we were writing this out, we'd have put a "1" in the 32s place.

We have five chips left. We don't have enough to make a stack of 16 or 8, but we do have enough to make a stack of 4, which leaves us with 1 left over. So we have a stack of 32 chips, a stack of 4, and a single chip. This corresponds to the binary word 100101. Same number of chips that we started with, just stacked differently.

Now let's convert to octal. An octal digit represents up to three binary bits; the maximum octal number of 7_8 is binary 111 . The quantity we just calculated, 100101_2, is six binary bits, so it will take two octal digits to represent the number, one for the most significant three bits and one for the least significant three. We can divide the bits into two groups of three:100 and 101. These represent two octal digits. 100_2 is 4_8 or 4 in the 8s place, and 101_2 is 5_8 or 5 in the 1s place.

Let's stack the chips. We have enough chips make four stacks of 8, which will leave 5 chips left over. So our octal number is 45_8. That is, 37_{10} = 100101_2 = 45_8.



Wow, what a lot of work! Why would you do such a thing? To get the number in a form that both humans and computers can understand, that's why. And if you bought a scientific calculator that converts among decimal, binary and octal, or if you have a scientific calculator app on your computer, you can let the machine do the work.

HEXADECIMAL NOTATION

It would be nice if we could use two decimal numbers to represent 8-bit words instead of 6-bit words, because the 8-bit byte is so common in computing. But that would mean each digit would have to be able to represent a 4-bit word, which is up to16 different values. An octal digit can only represent 8 values including zero, and a decimal digit can only represent 10 values including zero.

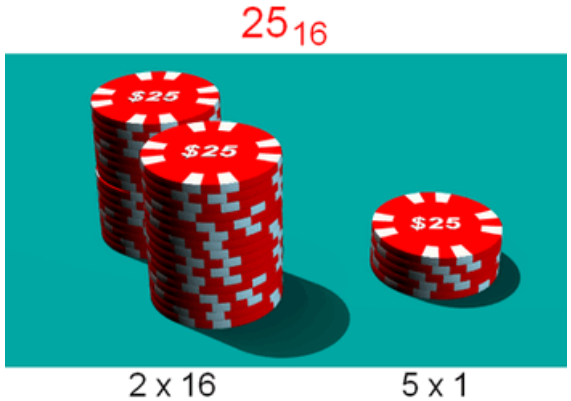
We can make a single digit represent four binary bits by adding six more possible values beyond “9.” The computer gurus have adopted the letters A through F to represent the six values 10_{10} thru 15_{10}. So we wind up with base 16_{10}.

0 1 2 3 4 5 6 7 8 9 A B C D E F

This is called the **hexadecimal** numbering system because it uses the ten decimal numbers plus six (“hex” as in hexagon) letters.

Hexadecimal is commonly abbreviated **hex**. Since a single hex digit can represent numbers to 15_{10}, the next digit to the left would be the "sixteens" place; that is, a "1" in that place would mean 16_{10}. An "F" in the sixteens place would represent fifteen sixteens, 15_{10} x 16_{10} or 240_{10}. The largest number that can be represented by two hex digits is FF_{16}, which is 240_{10} + 15_{10} = 255_{10}.

Let's restack our 37 chips. If the second digit from the right is the16s place, two stacks of sixteen chips will use up 32 of them, leaving five in the ones place. 37_{10} is 25_{16}.



Again you will ask – will I really run into hexadecimal numbers? All the time. Computer data and instructions are routinely coded in hex. Fortunately, if you've got a scientific calculator that will convert octal, it will also convert hex.

When writing a word like 10010110, the digits are often separated so: 1001 0110. These groups of four convert to two hexadecimal digits. And, as you know, a group of four bits is a nibble – that is, a hex digit is a nibble. So we really do use that word after all.

Decimal number	Hexadecimal number	Decimal number	Hexadecimal number
00	00	16	10
01	01	17	11
02	02	18	12
03	03	19	13
04	04	20	14
05	05	21	15
06	06	22	16
07	07	23	17
08	08	24	18
09	09	25	19
10	0A	26	1A
11	0B	27	1B
12	0C		
13	0D	all the way to...	
14	0E		
15	0F	255	FF

2-digit (two nibbles / one byte) hex values

Because computing normally uses 8-bit words (and multiples of 8 bits, like 16, 32 and 64 bits), octal notation is not used much any more; everybody works in hex. A hex number is often identified by a dollar sign, as in $4F.

BINARY CODED DECIMAL

Sometimes we do want to count in 10s. In these cases we use a 4-bit counter set up so it counts to 1001 (that is, 9_{10}), then resets on the next input pulse and puts out a carry signal to the next counter. This is known as **binary-coded decimal** or **BCD**: decimal numbers coded in binary notation.

BCD drives other binary circuitry crazy, but often there isn't any other binary circuitry. In a frequency counter, for example, we're just counting cycles, and all we want is for the frequency to show up on a display made of decimal digits. A series of BCD counters counts the cycles for a second, then another circuit freezes the count and transfers it to a display. (This also involves a circuit that translates BCD to whatever form needed by the display to show numbers that humans recognize. More on this in chapter 9.) The BCD counter then resets to all zeroes and starts the next count.

There's one other reason to use BCD. Binary math, for all its precision, has a problem. Computers often use something called **floating point** arithmetic where the position of the point moves depending on the size of the numbers. In floating-point arithmetic, small errors can creep in. Where absolute precision is needed, as in bank records, computing is sometimes done in BCD.

BEYOND NUMBERS

So much of what we do involves reading and writing, it is convenient to use binary words to represent letters. “But they're not letters, they're numbers!” I hear you say. They're only numbers if we say they're numbers. Let's step back and look at the broader picture: what binary data really is, is **a sequence of ordered 2-state values** – values that mean something. What they mean is up to us. Just because we can use bits to count, that doesn't mean we have to. They're just high and low voltages. They can represent whatever we want them to represent – numbers, letters, light bulbs, even patterns:

```
o1oooooo1oooooo1oooooo1oooooo1oooooo1oooooo
oo1oooooo1oooooo1oooooo1oooooo1oooooo1ooooo
ooo1oooooo1oooooo1oooooo1oooooo1oooooo1oooo
oooo1oooooo1oooooo1oooooo1oooooo1oooooo1ooo
ooo1oooooo1oooooo1oooooo1oooooo1oooooo1oooo
oo1oooooo1oooooo1oooooo1oooooo1oooooo1ooooo
o1oooooo1oooooo1oooooo1oooooo1oooooo1oooooo
```

The bit pattern above might be used to control a loom, weaving a zigzag pattern into a piece of cloth. In fact, as far back as 1801, paper cards with holes punched in them were used to do just that. Rolls of paper with holes punched in them have been used to play pianos. A machine that used

punched cards to record U.S. census information was the beginning of IBM. The first step in freeing computers to do things besides math is to realize that bits can represent anything. As long as we're still dealing with just two levels, we're still dealing with binary. In future chapters, we're going to look at some of those other things. Right now, let's just look at a few systems that encode letters into bits.

EBCDIC

EBCDIC stands for **extended binary coded decimal interchange code**. The first “C” is pronounced like “S”: EBB-sed-ick. EBCDIC is an **alphanumeric** code, meaning that it can encode both letters and numbers.

EBCDIC was invented by IBM in the 1960s for use in their large mainframe computers. At the time, a more elegant coding system called ASCII was already under development, but IBM was eager to introduce its model 360 computer in 1964, and either they didn't have time to implement ASCII, or ASCII wasn't ready.

EBCDIC uses 8-bit words, two hex digits, to denote a particular quantity. With 256 possible values (counting zero), there is plenty of room for the numbers 0-9, 26 upper-case letters, 26 lower-case letters, common punctuation symbols, and a small collection of instructions.

EBCDIC is a bit clumsy, and it would long since have disappeared but for the fact that the model 360 computer was a smash hit, firmly imbedding EBCDIC into standard computer practice. Certain IBM computers support EBCDIC even today, because there is so much archival data coded in EBCDIC and because there are so many EBCDIC process and control programs that work flawlessly. The people who wrote that stuff have passed on, taking their knowledge with them, so modernizing those programs would mean starting over from scratch.

Our “bits can be anything” approach has already taken us beyond numbers and letters; in EBCDIC, some combinations of bits represent instructions. For example, there's a command that tells the system to ring a bell to alert the user.

ASCII

EBCDIC was a way station on the road to **ASCII**, which is short for **American Standard Code for Information Interchange**. It is an extension of older telegraph and teleprinter codes. Work on ASCII was begun in 1960 by what is now the **American National Standards Institute** or **ANSI**. It was published in 1963 and it's had updates since.

ASCII is 7 bits, one bit short of a byte; thus it can encode only 128 characters. Of these, 95 are printable characters (including space, which is considered a blank character). The other 33 are instructions, mostly for formatting, which now have largely been superseded by other formatting methods. ASCII is the standard system for representing numbers and the English alphabet in binary code. Unlike EBCDIC , which was an in-house system for IBM, ASCII is used by everybody, permitting free exchange of data among computers built by different manufacturers.

Just for reference, there's an ASCII chart in the review chapter. You don't need to memorize it.

UTF-8

ASCII was developed by Americans for the U.S. computer industry. As the rest of the world began to adopt computers, and especially as the world began to use the Internet, the USA-specific nature of ASCII became a problem. In the world data community, American ASCII is now considered a subset, called **US-ASCII**, of a broader code.

The communication gurus of the world have been working on a **universal character set** or **UCS** that currently has tens of thousands of characters, and will eventually include all the characters used by every language. **UTF-8** is an 8-bit standard that includes many of those characters.

UTF-8 is backward-compatible with ASCII; that is, all the ASCII codes are part of UTF-8. Any time the 8^{th} bit is 0, the other 7 bits represent a traditional ASCII character.

In the last few years, UTF-8 has begun to replace ASCII as the standard format for the Internet. There are also UTF-16 and UTF-32 versions, but these are not likely to become as widely used, at least in the near future, because they are not ASCII-compatible.

LARGE BINARY NUMBERS

When we get to numbers with many bits, it's customary to round off the numbers when we speak of them, kind of a verbal shorthand. For example, the number of memory addresses in a computer is a power of 2, but we quantify it in decimal terms.

- 10 bits represent $1{,}024_{10}$ values, but we call this “1k.”
- 16 bits represent up to $65{,}535_{10}$ values. We call this “64k.”
- 20 bits represent up to $1{,}048{,}576_{10}$ values. We call this “1 meg.”
- We call 1,024 megs “1 gig.”
- We call 1,024 gigs “1 ter.”

Whether we mean, say, 1 gibabit or 1 gigaByte depends on the situation. More on this in the computer chapter.

So: binary can count, it can talk, and it can issue instructions. Can it think? In a sense it can, as we'll see in Chapter 3.

End of Chapter 2

EM4 Chapter 3
BOOLEAN LOGIC

Objectives of this chapter:

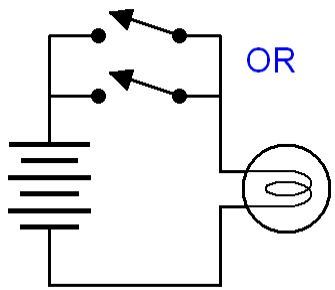
- *define "Boolean logic" and discuss the basic concept*
- *discuss AND, OR, NAND, NOR, EXCLUSIVE OR and NOT functions, citing familiar real-world examples*
- *show the schematic symbols and truth tables for each of these gates*
- *discuss tri-state outputs and their function*
- *explain single, dual, triple, quad, hex and octal IC packages*
- *define "enable," "active low," "race state," "wired OR."*

In the early and middle 1800s, mathematician George Boole developed a formal system of logic based on two states: something is true, or something is not true. He discovered that any true-false situation involving one or two factors can be described by one of three possibilities that he called AND, OR, and NOT.

- **NOT**: If A is true, then B is false.
- **OR**: If A or B is true, or if both are true, then C is true. If A and B are both false, then C is false.
- **AND**: if A and B are both true, then C is true. If either A or B is false, or if both are false, then C is false.

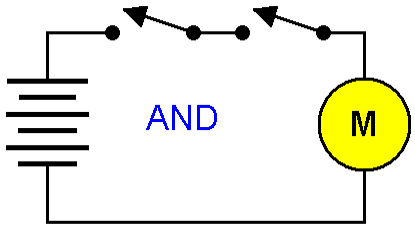
This is called **Boolean algebra** or **Boolean logic**, also **combinational logic**. Boole intended his system as a method of thinking. He believed philosophical questions were not fundamentally philosophical but mathematical. He thought Boolean logic could solve problems from mathematics to marriage disputes. (He was at least partly right about both.) However, it is doubtful that he envisioned binary computers. In Boole's time, and as late as the 1940s, people assumed computing machines would count in tens, the way people count. But even before binary computers, people were using Boolean two-state logic in electrical circuits without realizing it.

You've got a Boolean OR built into your car. If the driver's side or passenger's side door is open, or both, the dome light comes on. If neither is open, the dome light is off.



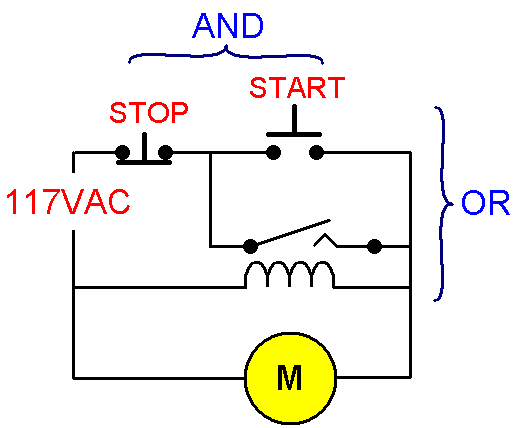
I toured a commercial bakery as a kid, back in the days when safety devices were crude. The bakery had a dough mixing machine the size of a UPS van, and it was often necessary to operate the machine while the door was open. So, there were two pushbuttons, one on either side of the door, five feet apart, and the machine only ran as long as

both buttons were pushed. That meant you needed both hands to hold the buttons, so you had no opportunity to stick your arm into the machine. That's a Boolean AND.



In Las Vegas casinos, it takes two keys held by two different people to get into the counting rooms. Another AND.

Many machines are started by a pushbutton that pulls in a relay. Once pulled in, the relay keeps itself pulled in via its own contacts. (That is, it **latches**.) Pushing a normally-closed button that's wired in series with the system stops the machine. So, for the machine to run, the start button OR the relay contacts or both, AND the stop button, must be closed.



These examples are so familiar that it's easy to miss their significance: they are more than circuits, they are thoughts. They monitor conditions and make decisions, behaving within their limited range the way a human would. Boolean logic gives us a way to outsource mental functions, to cause decisions to be made without having to make them ourselves.

TRUTH TABLES

In Boolean logic, each factor that contributes to a decision is treated as an input, identified by letters A, B, C and so on, which in Boolean are supposed to be capitalized. (Not everybody does.) The the logical result is treated as an output, labeled “Y.”

The usual way to represent all the possible input and output states of a Boolean decision is a **truth table**, which lists every possible combination of input states and what the resulting output state will be for each. The truth table to the right is for the AND function, where both inputs must be true for the output to be true. Study it until you've got a secure grasp of it before reading further.

AND

input A	input B	result
true	true	true
true	false	false
false	true	false
false	false	false

BOOLEAN SWITCH CIRCUITS

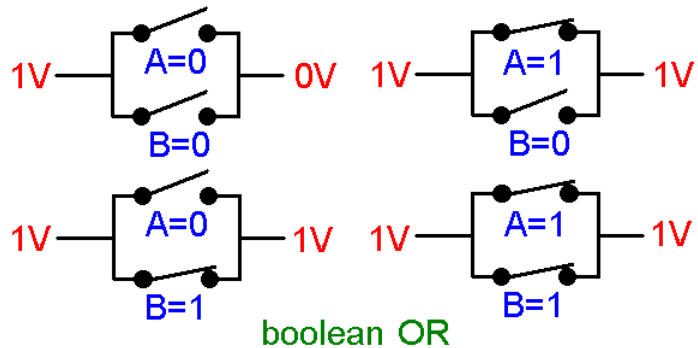
When we think of Boolean functions as switch circuits, the natural thing is to think of situations where a voltage at a particular input either does or does not get through to the output, depending on whether the switches are open or closed. This may be one reason why a Boolean output is labeled "Y." In electricity and electronics, "Y" is admittance, which is the inverse of impedance:

$$Y = 1 / Z$$

An open switch has infinite impedance and zero admittance; a closed switch has (ideally) zero impedance and infinite admittance. In the context of current flow, infinite admittance means a voltage ("1") can be transmitted to the output.

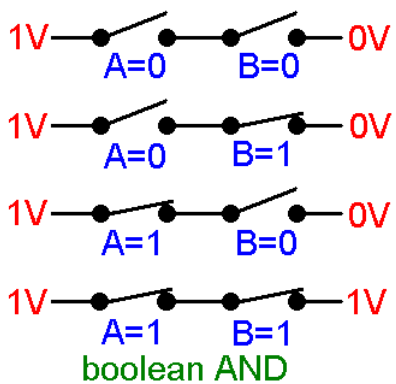
OR

In a Boolean OR, the A and B inputs are switches wired in parallel as shown above right. A voltage representing "1" is connected at the left – for convenience, we might as well use one volt. If both switches are open, the voltage is not transferred to the output on the right, which stays at 0V. However, if either switch is closed, or if both are closed, the output goes to 1V.



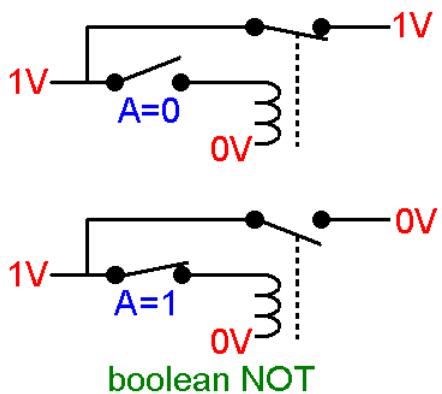
AND

In a Boolean AND, the A and B inputs are switches wired in series. If one or both switches are open, the voltage is not transferred to the output, which stays at 0V. However, if both switches are closed, 1V will appear at the output.



NOT

A Boolean NOT is an inversion, like a switch connected to the coil of a relay with normally closed contacts. When the switch (A) is closed, which is "true" in Boolean, putting 1V across the coil, the relay contacts open, leaving the output at 0V. When the switch is open, 0V appears across the coil, the relay contacts close, and 1V appears at the output.



By combining the three basic Boolean functions, we can build a system that performs any logical operation of any complexity.

The first computers to do Boolean calculations actually did use switches and relays; for a long time it was not obvious that electronic circuits were better. For one thing, the only available electronic switching device until the 1950s was the vacuum tube, and tubes are much better in analog applications than digital ones. Tubes also have a much shorter life expectancy than good relays. It was only after someone pointed out that "life expectancy" should be measured in terms of the number of operations a switch can perform before it fails (in the case of tubes, almost unlimited) that computers became electronic.

BOOLEAN AND BINARY

2-state Boolean logic and 2-state binary math are natural partners, but they are not the same. For one thing, in Boolean logic there is no such thing as a subtraction or division operation. Binary is a way of counting; Boolean is a way of thinking.

However, the two have enough in common that, if we say a binary "1" means "true" and "0" means "false," and if we use a voltage to represent "1" and no voltage (zero volts) to represent "0," we can build a machine that connects the world of electronics with the world of logic – a machine which has the electronic equivalent of thoughts.

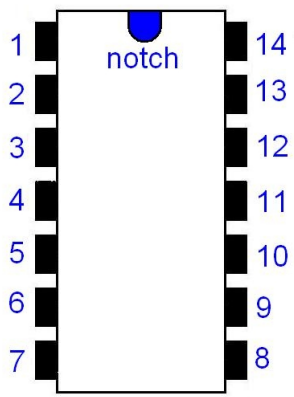
GATES

An electronic Boolean decision-making circuit is called a **gate**, not to be confused with the gate of an FET. Gates are named for the functions they perform. Just as a resistor, NPN transistor, P-channel FET etc have standard schematic symbols, each type of gate has a symbol. As we've seen, Boolean functions can be done with switches; electronic gates are not required. However, if we use electronic gates, we can control Boolean functions electronically instead of mechanically.

This is the starting point for a machine that can solve a complex logic problem, often as well as a human could solve it and much more quickly.

Gates were once built with tubes, then discrete transistors. Today they're built in integrated circuits. Many ICs contain several gates of one type. These are usually referred to by the following names:

single	**one gate to a package**
dual	**two gates to a package**
tri-	**three gates to a package**
quad	**four gates to a package**
hex	**six gates to a package**
octal	**eight gates to a package**



The traditional gate packages are 14-pin and 16-pin DIP ICs, and their smaller surface-mount equivalents; the pins are arranged in two rows of 7 or 8 pins. Looking at the IC from the top, pins are counted starting with the upper left pin and going counterclockwise; a notch or hole at one end of the chip identifies the starting point. In most digital chips, the lower left pin (7 or 8) is ground, often called $\mathbf{V_{EE}}$ for TTL chips and $\mathbf{V_{SS}}$ for CMOS, and the upper right pin (14 or 16) is +DC, often called $\mathbf{V_{CC}}$ for TTL and $\mathbf{V_{DD}}$ for CMOS.

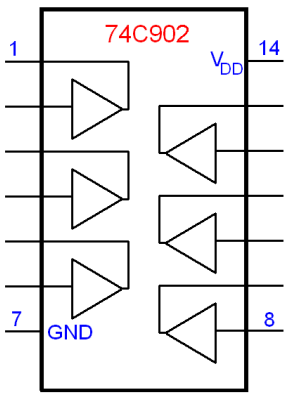
As we look at various gates, I'm going to mention some standard IC numbers. Many of these are considered obsolete, not because their functions are obsolete but because the functions are now usually part of computer programs. Numbers I've quoted like “74C14” refer to the earliest CMOS equivalents of TTL circuits. Nowadays you'll find more letters in the middle of the numbers, e.g. 74HCT14 or 74ACT14, which refer to certain operational characteristics as noted in chapter 1.

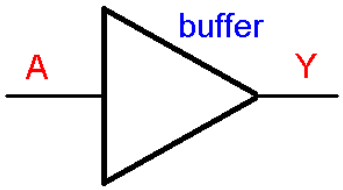
BUFFERS

Strictly speaking, the two simplest gates are not really gates, because they don't have a decision-making function. However, they are physical versions of real Boolean functions, and they work right along with the “legitimate” gates to create Boolean systems.

NON-INVERTING BUFFER

A non-inverting **buffer** is simply a repeater: if the input is 0, the output is 0; if the input is 1, the output is 1. In the truth table, Y=A.

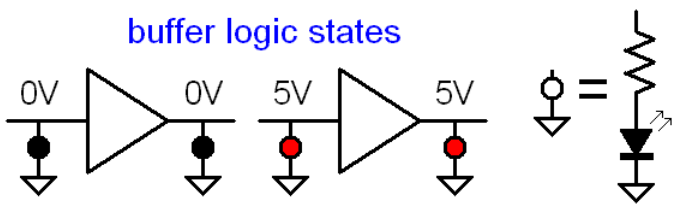




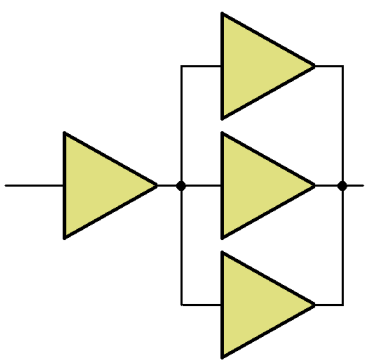
buffer

A	Y
1	1
0	0

The schematic for a non-inverting buffer is a triangle, same as for an amplifier. An example is the 74C902 hex buffer, six buffers plus DC and ground pins.

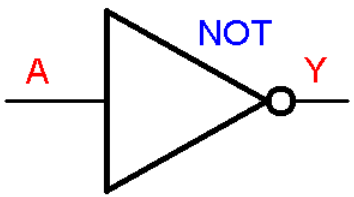


A buffer has no real Boolean function, but it has a practical electrical function. Buffers can be used to isolate a gate from a load that might be too heavy for it. They can even be connected in parallel to obtain more drive current. This may cost nothing, as a hex or octal buffer IC often winds up with a few extra buffers that can be used for this.



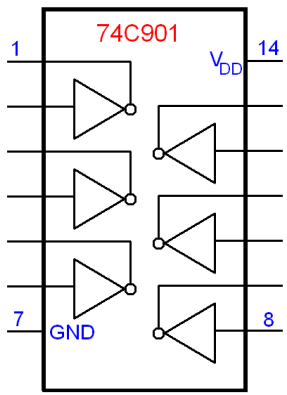
NOT (INVERTING BUFFER)

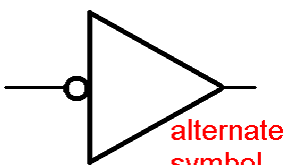
The simplest real Boolean function, NOT, means if A is true, Y is false, or if A=1, Y=0. Electrically, that's an inverter. The symbol is a buffer with a little circle at the input or the output. **This kind of circle always signifies an inversion**. An example would be a 74C901 hex inverter.

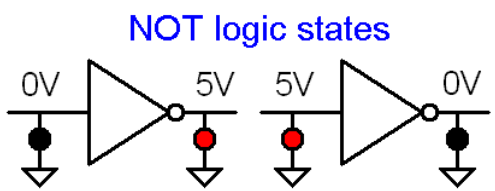


NOT

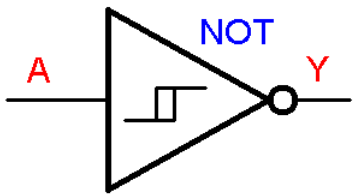
A	Y
1	0
0	1







SCHMITT TRIGGER INVERTING BUFFER

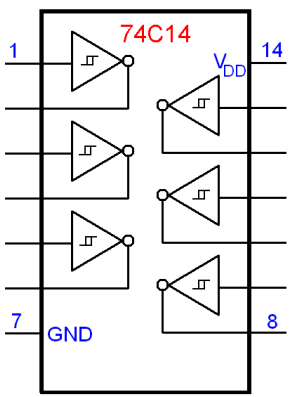


NOT

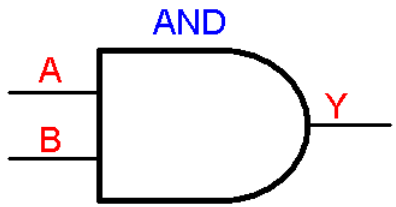
A	Y
1	0
0	1

A Schmitt buffer uses positive feedback to clean up and speed up up transitions. This means it has hysteresis; the input must go past the halfway point in each direction for for the output to go high or low. If you were to make a graph of input versus output voltage, you'd get the S-like shape you see inside the symbol.

An example would be the 74C14, a hex Schmitt trigger inverter. With a 5V supply, the input voltage must rise to about 3.5V for the output to go to 0, and then fall below about 1.7V for the output to go to 1. Besides being fast and having a clean, sharp rise and fall, a Schmitt buffer greatly reduces the shoot-through problem. The output is always 1 or 0, never undefined.



AND

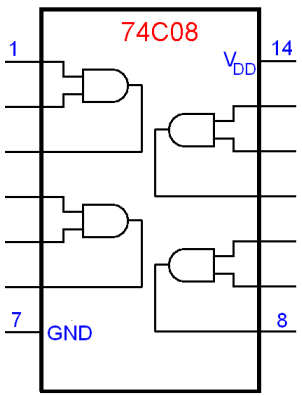


AND

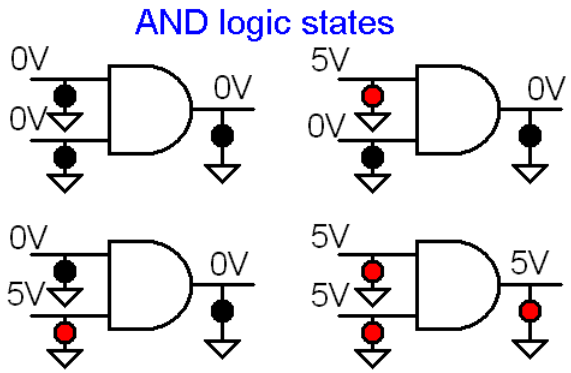
A	B	Y
1	1	1
1	0	0
0	1	0
0	0	0

The AND symbol has a flat input end and a rounded or arch-shaped output end. There are always at least two inputs, as in the 74C08 quad 2-input AND. Some ANDs have up to eight inputs.

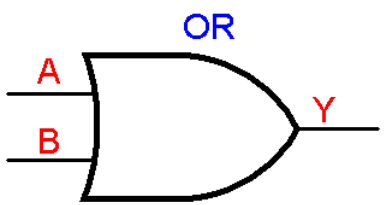
With an AND gate, the output is 1 if all inputs are 1. If any input is 0, the output is 0. You could call this an “all or nothing” gate.



In modern cars, to start the engine you must turn the key to “start” and step on the brake at the same time. That's a Boolean AND.

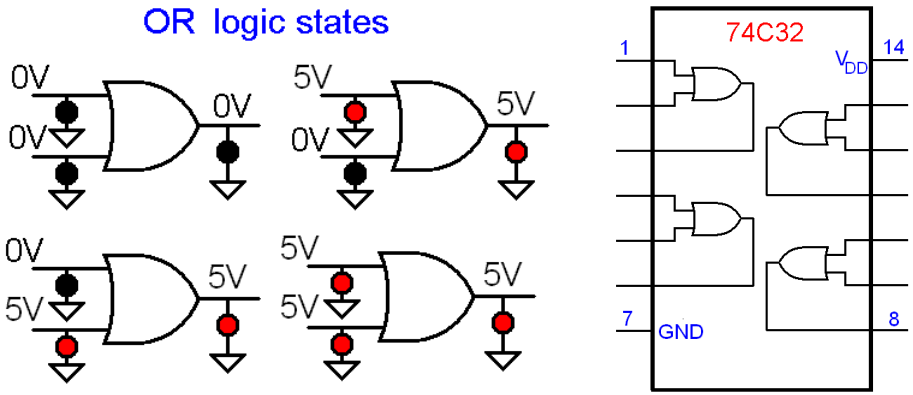


OR



A	B	Y
1	1	1
1	0	1
0	1	1
0	0	0

With an OR gate, the output is 1 if any input or all inputs are 1. If all inputs are 0, the output is 0. I've mentioned the dome light arrangement in your car, a 2-input OR function, or a 4-input OR if the back doors turn on the dome light too. The symbol is similar to the AND gate symbol, but the input end is curved and the output end peaked to signify that it's an OR. You could call the OR an “any or all” gate. An example is the 74C32 quad 2-input OR.

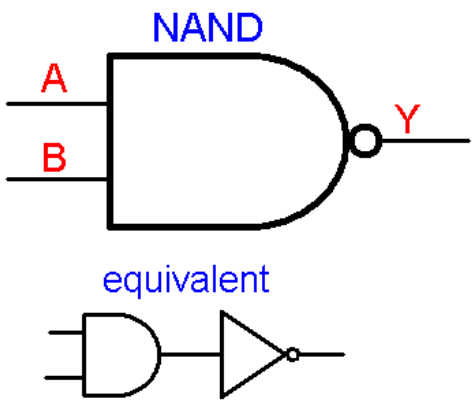


Almost any logic system can be built with ANDs, ORs and NOTs. However, some functions involve complicated assemblies of those basic gates. Things get simpler if we expand our library of gates a bit.

NAND

Putting an "N" on the name of a gate signifies that the function is inverted. “NAND” is short for “NOT AND.” The NAND function is the same as the AND function, except that the output is inverted; when all inputs are 1, the output is 0.

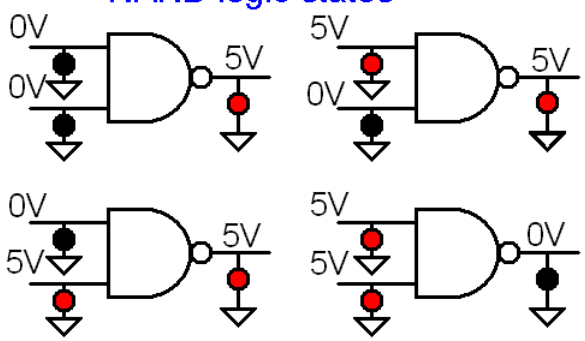
The symbol is an AND symbol with the little circle added.
Functionally, this is the same as an AND followed by a NOT.



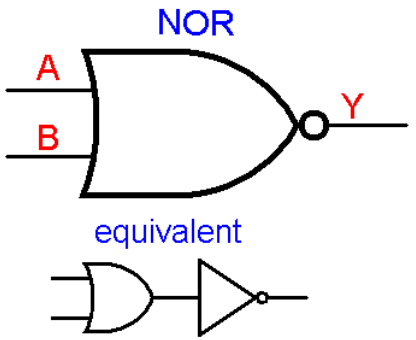
NAND

A	B	Y
1	1	0
1	0	1
0	1	1
0	0	1

NAND logic states



NOR

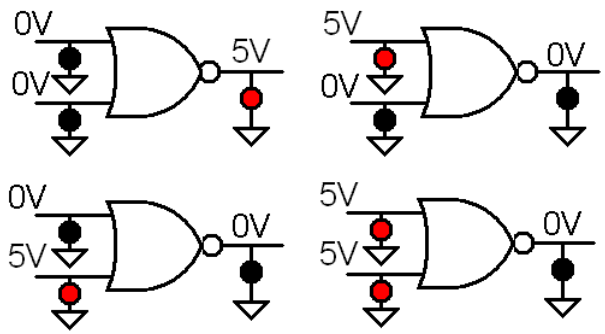


NOR

A	B	Y
1	1	0
1	0	0
0	1	0
0	0	1

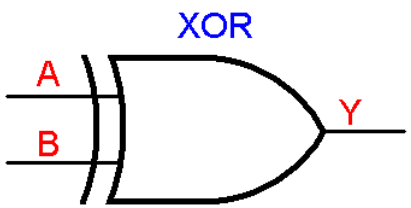
The NOR gate is an OR gate plus an inversion, signified by an OR symbol with a circle at the output.

NOR logic states



XOR

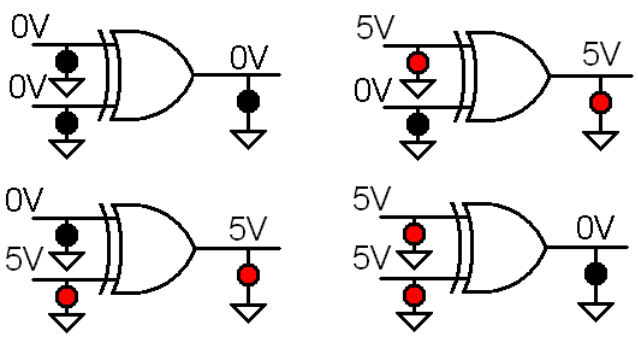
XOR is short for **exclusive OR**. It always has only two inputs. If both inputs are 0, the output is 0. If either input is 1, the output is 1. Up to this point an XOR is the same as an OR. But if both inputs are 1, the output is 0. That's why it's exclusive. You could call it an “any but not all” gate.



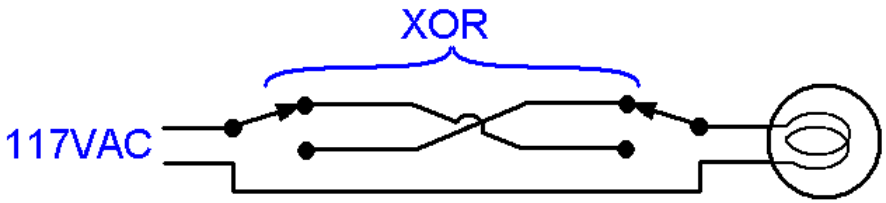
XOR

A	B	Y
1	1	0
1	0	1
0	1	1
0	0	0

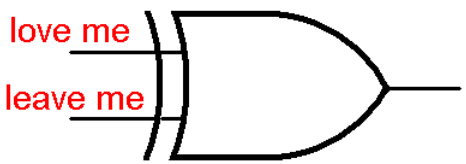
XOR logic states



A pair of 3-way light switches at either end of a hallway is an XOR. If both toggles are down, the light is off. If either toggle is up, the light is on. If both are up, the light is off. Take your finger and trace the current flow.



Another XOR:

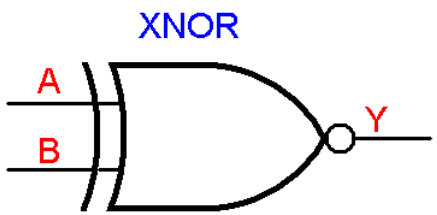


love	leave	OK
1	1	0
1	0	1
0	1	1
0	0	0

Hey, it's 3:00 AM, I'm getting punchy.

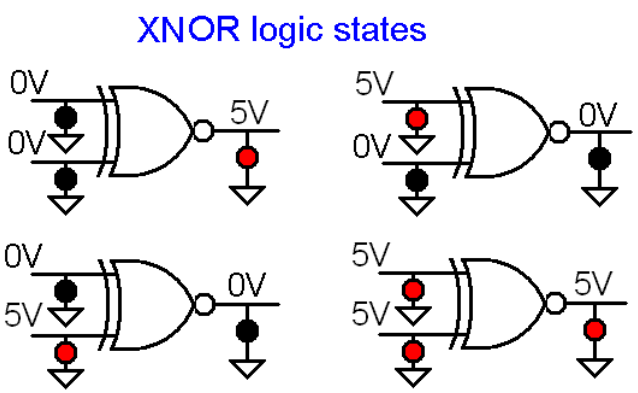
XNOR

An XNOR, as you could probably have guessed, is an XOR with an inversion at the output.



XNOR

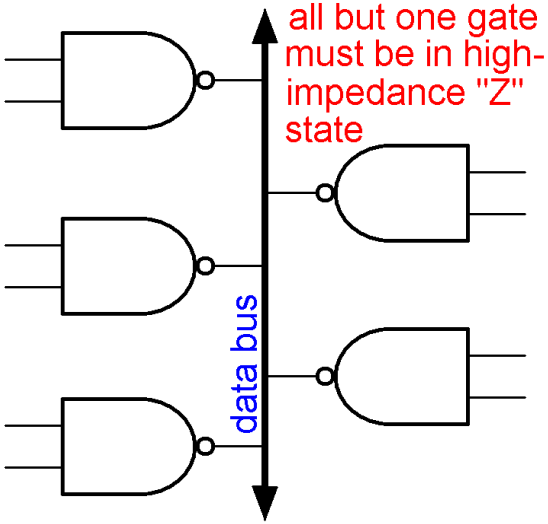
A	B	Y
1	1	1
1	0	0
0	1	0
0	0	1



TRI-STATE OUTPUTS

Some gates have three output states:

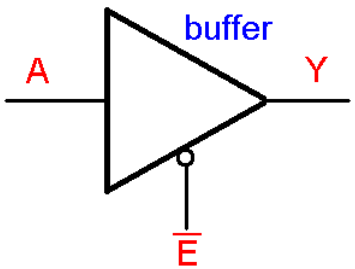
- **1**, where the output is connected to +DC (pull up)
- **0**, where the output is connected to ground (pull down)
- **Z**, where the output is disconnected (open circuit)



This is useful where, for example, several circuits share a common communication channel called a **data bus**. If all the outputs are connected to the bus all the time, some will try to drive it to 1 and some will try to drive it to 0. This is called a **bus fight** or **collision**. We avoid it by making all the outputs **tri-state** outputs, and whenever one of them is sending 1's and 0's, all the others are in their high-impedance Z state, effectively disconnected, politely waiting for their turn to use the bus.

ENABLE INPUTS

An **enable** input controls whether a tri-state output is in active or high-impedance mode. The schematic symbol below shows a buffer with a control input labeled nE (NOT E), meaning the buffer is active when the enable is low, and disconnected when it's high. (This is an example of an active-low function mentioned in chapter 1.) In the truth table, X means “don't care” because, when the output in its high impedance state, the input state is irrelevant.



buffer

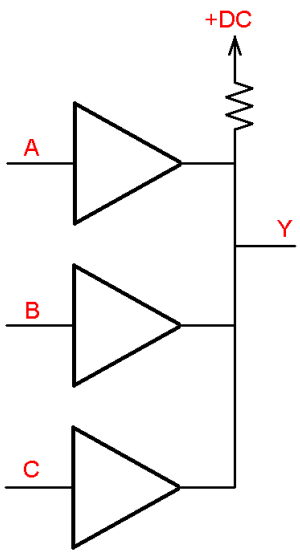
$\overline{E}$	A	Y
0	1	1
0	0	0
1	X	Z

I want to call your attention to something: note that there's a line over the “E.” **In Boolean, that's how a NOT is indicated**. In the case of an enable input, the line means that control input is active-low.

Many data buses have 8 separate wires to be able to pass a full byte at a time. Accordingly, some tri-state buffer chips are organized as octal buffers that are all put into operating or high-impedance mode by a common enable pin. Several of these chips could be paralleled for, say, a 64-bit bus.

OPEN COLLECTOR / DRAIN

Another way of avoiding bus fight is to use gates that pull down, but do not pull up; their pulldown transistors have **open collectors** or **open drains**. These can be directly wired in parallel, along with a common pullup resistor. Each output rests in its high state when it is not sending data. When any output goes low, the common output bus goes low. This is called a **wired AND** circuit because the common output is only high when all the individual outputs are high. In active-low circuits, where low = 1, this becomes a **wired OR** circuit, where any individual output that goes low pulls the common output low. There is no electrical difference; it's just a matter of how you look at it.

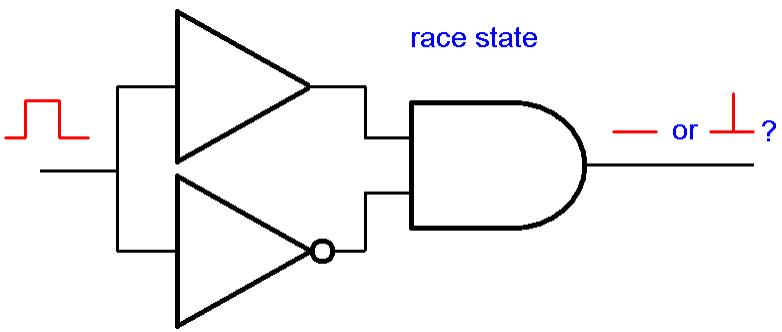


An open collector/drain circuit can also act as a level shifter; the gates' DC supply can be one voltage while the output is connected into a circuit with a different supply voltage. An example is the 74C906, a non-inverting hex buffer whose open drains can handle up to 18V.

GATE DELAY

I've been implying that Boolean gates do their work instantly, but of course that's not strictly true. It takes a few nanoseconds for a gate to react. In a system where a lot of data flows through a lot of gates, this puts a limit on the speed. The input-output time is called **gate delay** or **propagation delay**.

In poorly designed circuits, there's often a question about which of two things will happen first. This is called a **race state**: there's a race to see which of two events will occur first, and the outcome is uncertain.

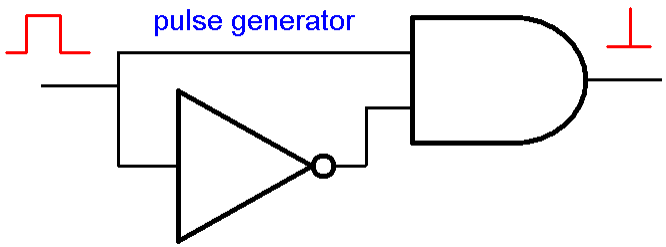


For example, in the diagram above, whether the input is low or high, one of the inputs to the AND will always be low, so the output will be low. But, during the transition, could there be a moment when both AND inputs are high? That depends on whether the buffer or the NOT has a longer gate delay, and who knows what the answer might be – especially if they're in two different chips made by two different manufacturers.

In a well designed circuit, each event is the result of a specific previous event, so the sequence of events can't get out of order. However, not all circuits are well designed. You might find that, after replacing a defective IC with a supposedly identical part, the circuit doesn't work. If you can't find any other reason, you might want to closely examine the schematic to see if a race state exists.

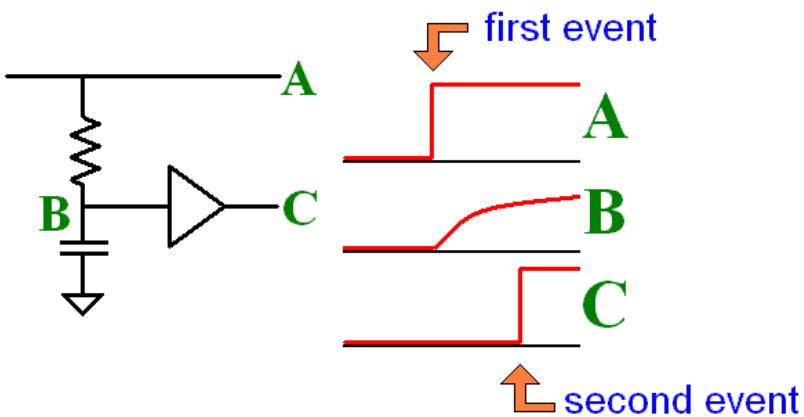
Sometimes a gate delay can be useful. For example, many circuit functions require an **edge trigger**, which causes something to happen at the moment when a clock transitions from 0 to 1 or 1 to 0. It is important that just one single thing happens at that time, like a turnstile gate that lets one person through at a time. If the trigger lasts long enough for several events to sneak through, that's a malfunction.

The diagram below is a circuit often used to create a short trigger pulse. This is not a race condition; there is no question how the circuit will behave. We can be quite certain it will take longer for the clock signal to reach the lower AND input via the NOT gate than it takes to reach the upper input via the direct path.



On a positive clock transition, the upper AND input will go high before the lower AND input goes low, so there will be a moment when both inputs are high. This will cause a positive spike at the AND's output whose time duration is about equal to the gate delay of the NOT.

Note that this circuit will only work if the input clock has a virtually instantaneous rise time. Not all digital systems require sharp, square-cornered voltage transitions, but many do, especially the computer-based systems we're going to meet in the following chapters. Clean waveforms with well-defined corners should be considered a basic requirement of good digital practice.



WAVE SHAPING

Sometimes we'll want to do a little fiddling with time delays. For example, we may want to be sure something happens in a specific order – no race states. This can be done by taking a logic transition, stretching it out via an RC time constant, then squaring it off again using a buffer. Another way of doing this is to send the logic transition through a series of buffers, which are easier to put on a chip than a resistor and a capacitor. Or, we may be starting with a waveform that may not have sharp transitions to begin with. In each case, we can use wave shaping techniques to square up the transitions. Schmitt-trigger gates work especially well for this.

BOOLEAN MATH

When we do Ohm's Law calculations, we don't draw resistors and we don't draw resistor symbols. We write “R” and let it represent the resistance. We can do the same thing with Boolean logic: we can use letters and symbols to represent the functions.

In this way, we can build a system of mathematics that is different from number math, but uses a lot of the same principles. Like number math, Boolean math lets us design systems by moving symbols around rather than soldering parts together. This has three benefits.

One, the rules let us check for errors and correct them a whole lot more easily than modifying circuitry.

Two, the rules often show us how to simplify a design. This means we can build a circuit with fewer gates, which takes less time and materials and should work more reliably. And when we're not using gates, when we're using relays and valves and such, the savings can be significant.

Three, computers don't just crunch numbers, especially in machine control applications. They do Boolean too. Knowing Boolean math is handy, and often necessary, when writing programs.

You won't get all of the following on the first read, but if you keep coming back to it, you'll get it eventually, piece by piece. None of the individual concepts is rocket science.

Where will you use this stuff? (Aside from passing the ETA exam, that is.) That's anybody's guess. I just went back over my notes from when I started writing these study guides a couple of years ago, and the landscape has already changed in ways nobody predicted. I've always found that having as broad a knowledge base as possible is a good way to never be without a job offer. Even if you don't wind up *doing* Boolean math in your job, the fact that you *know* Boolean math will raise you a notch in the eyes of your employer.

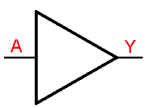
WRITING IN BOOLEAN

Truth tables can be written as equations. We can refer to the result as "Y," the way we do with truth tables, or choose another letter. For the simplest functions, i.e. inversion and noninversion, we can use the same letter:

$\mathbf{A = A}$ (output equals input, in other words, noninversion.)

A Boolean statement refers to a function, and also to the gate that performs that function.

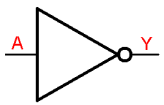
Noninverting Buffer: a simple equal sign joining the A input to the Y output.



$\mathbf{Y = A}$ or $\mathbf{A = A}$

A Schmitt circuit would have the same equation, since the trigger is an electrical function, not a logical one.

NOT: in Boolean, inversion is represented by a line over a letter. The equation for an inverter is:



$$Y = \overline{A}$$ (Y is notA, the opposite of A)

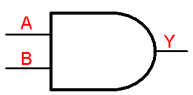
As you've seen, the line also shows up in gate diagram symbols; for example, an enable input identified by an “E” with a line over it means the line must go to “0” to enable that function.

NOTs can be stacked. Three NOT gates in series would be:

$$Y = \overline{\overline{\overline{A}}}$$

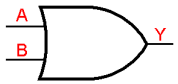
Four stacked NOTs wouldn't be an inversion. If you flip something over four times, it's right side up again. In complicated Boolean systems, that kind of thing happens a lot.

AND: a dot.



$$Y = A \bullet B$$

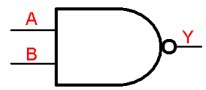
OR: a “plus” sign.



$$Y = A + B$$

Confusing as all get-out, that says “and,” right? But in Boolean equations, the plus symbol means OR. There's a reason for it, which we'll get to shortly.

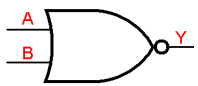
NAND: as you'd guess, the AND equation with a line over A and B.



$$Y = \overline{A \bullet B}$$

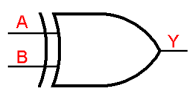
Note that the AND is performed first, then the result is inverted, as in a NAND symbol with the circle at the output. Rule: when you see a line over more than one symbol, the inversion is done last.

NOR: same as OR, with a line over it.



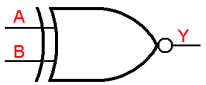
$$Y = \overline{A + B}$$

XOR: OR sign in a circle.



$$Y = A \oplus B$$

XNOR: XOR with a line, natch.



$$Y = \overline{A \oplus B}$$

Sometimes, people will use other symbols for these operations:

A and B: A•B, AxB, A^B, A*B, AB

A or B: A+B, A∨B

negation: $\overline{A}$, -A

BOOLEAN MATH RULES

If we try to do Boolean according to regular number rules, we'll get a hitch in our giddyup, cuz they're not all the same. But most of them are the same, and the ones that aren't make sense if you think about them. The main thing to remember is that Boolean is *not* a counting system; we don't do things like carrying and borrowing.

I can explain now why the symbols are a little weird. The AND function

$$Y = A \bullet B$$

uses a dot because **AND is a multiplication operation**. If we write out the truth table as a multiplication table, we'll get the same four equations we'd get with number arithmetic:

0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

In number math, people tend to omit the dot or the x unless it's needed to avoid confusion. It's legal to do the same thing with a Boolean AND:

$$Y = AB$$

The OR function

$$Y = A+B$$

uses a "+" symbol because **OR is an addition operation**. If we write out the truth table as an addition table, three of the equations will be the same as with number arithmetic (next page):

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 0$$

Only the last equation is screwy:

$$1 + 1 = 1$$

In binary, the result would be 0 and we'd carry a 1 to the next place. But in Boolean, the equation simply says that if either input is true, the output is true, and if both inputs are true, the output is doubly true, or as Tweety would say, it's twue, it's twue.

Now that we're thinking in math terms, we can develop rules about moving things around like we did with algebra. The last time we talked about algebra was EM1, so we need to go through this again. Most of the rules are exactly the same, though they are stated in Boolean terms. Each rule has a name, which I'm going to tell you. You don't have to memorize these names, but if you do I think you'll find the functions easier to remember, because the names describe the functions. There's a printout in the review chapter.

The **commutative** rule: the terms on either side of the equal sign can trade places.

$$(A \bullet B) = (B \bullet A)$$
$$(A + B) = (B + A)$$

commutative rules

The **associative** rule: in an equation with three ANDed or ORed terms, it doesn't matter which operation you do first. Any two can be the romantically involved pair and the other the third wheel; it doesn't matter who associates with whom.

$$(A \bullet B) \bullet C = A \bullet (B \bullet C)$$
$$(A + B) + C = A + (B + C)$$

associative rules

The **distributive** rule: if you have several terms inside parentheses and you want to multiply the whole thing by another term, multiply each separately. In other words, do an operation that distributes the terms inside the parentheses.

$$A \bullet (B \bullet C) = (A \bullet B) \bullet (A \bullet C)$$
$$A \bullet (B + C) = (A \bullet B) + (A \bullet C)$$

distributive rules

Those three rules work with both addition (OR) and multiplication (AND), and they're the same as the arithmetic rules that go by the same names. You're probably saying "meh," but you need to look at them closely until you can see the logical processes.

Here's another rule: when your brain is so full it's about to explode, don't keep trying to plow through this material. You may be able to learn it all at once, and your brain may still have it tomorrow, but you'll find it's all jumbled together and you can't remember what goes with what. If you've gotten to this point in one sitting, it's long past time to hang it up. Pizza!! Now!!!

* * * * * * * * *

Oh hello again, let's get back at it. Before you go on, go back to the beginning and skim over the stuff up to this point.

Next, we're going to follow the three rules above to some rules that pertain only to Boolean logic. Some of them are stupid-simple, but we like simple, right? Like a Swiss army knife, there's nothing special about the individual tools. But when they're put together, you have a tool for anything you might run into while working in Boolean.

The **identity rule**: two of something (or 20,000) are the same as one.

$$A \bullet A = A$$
$$A + A = A$$

identity rules

Check it out:

$$1 \cdot 1 = 1$$
$$0 \cdot 0 = 0$$
$$1 + 1 = 1$$
$$0 + 0 = 0$$

The **complementation rules**: A-AND-not-A = 0, A-OR-not-A = 1.

$$A \bullet \overline{A} = 0$$
$$A + \overline{A} = 1$$

not-identity rules

Make yourself a mental picture of series and parallel switches and you'll see that these two rules are always valid.

The **redundancy rules**, next page: these follow from the identity rule, stating that when there are several terms, they often just say the same thing several different ways, and we can simplify the equation by eliminating some of them. This is handy if it means we can eliminate gates and very handy if we can eliminate, say, $100 industrial-grade limit switches.

$$A + (A \bullet B) = A$$
$$A \bullet (A + B) = A$$

redundancy rules

Let's look at the first equation. B can be 1 or 0. If A = 1, there are two possibilities:

$$1 + (1 \cdot 0) = 1 + 0 = 1$$
$$1 + (1 \cdot 1) = 1 + 1 = 1$$

If A = 0, there are two possibilities:

$$0 + (0 \cdot 0) = 0 + 0 = 0$$
$$0 + (0 \cdot 1) = 0 + 0 = 0$$

In either case the answer is equal to A. Note that we could have used the words AND and OR, but we didn't have to. This is a pure addition and multiplication, the same as number arithmetic except that in Boolean arithmetic 1 + 1 sums to 1.

Let's look at the second equation. Again, B can be 1 or 0. If A = 1, there are two possibilities:

$$1 \cdot (1 + 0) = 1 \cdot 1 = 1$$
$$1 \cdot (1 + 1) = 1 \cdot 1 = 1$$

If A = 0, there are two possibilities:

$$0 \cdot (0 + 0) = 0 \cdot 0 = 0$$
$$0 \cdot (0 + 1) = 0 \cdot 1 = 0$$

Again, pure arithmetic.

Note that in each case there's a redundancy that we could reduce to A without even going thru the math. If we add or multiply using the distributive principle, we'll find A + A or A · A, which in either case equals A. (Don't you just hate it when I hide the candy like that.)

A more involved **redundancy** rule: in either the sum of products or product of sums equation below, the output = A.

$$(A \bullet B) + (A \bullet \overline{B}) = A$$
$$(A + B) \bullet (A + \overline{B}) = A$$

another redundancy

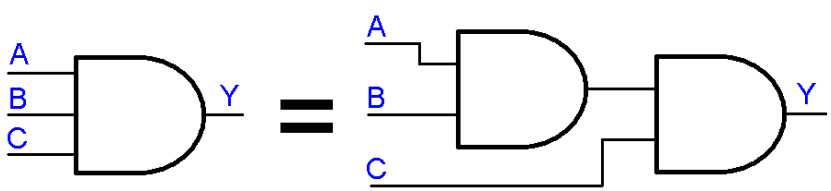
Guess what, I'm going to leave this one for you to work out the proof. Okay, I heard that! All right, maybe you didn't say it, but you were thinking it pretty loud. Shame on you. C'mon, you can do it.

ALTERNATE EQUATIONS AND SIMPLIFICATION

Boolean equations often reveal that there are several ways to accomplish a certain thing. For example, the following equations are essentially the same:

$$Y = A \bullet B \bullet C \qquad Y = (A \bullet B) \bullet C$$

The associative rule says the right-hand equation is functionally the same as the left-hand equation.
Now, there actually is such a thing as a 3-input AND; the 74C10 has three of them. But suppose you only have a 74C08 quad 2-input AND chip on hand. The equations above say you can build an equivalent circuit. And sure enough, that turns out to be true.



Here are two more rules:

$$\overline{A \bullet B} = \overline{A} + \overline{B}$$
$$\overline{A + B} = \overline{A} \bullet \overline{B}$$

deMorgan's theorems

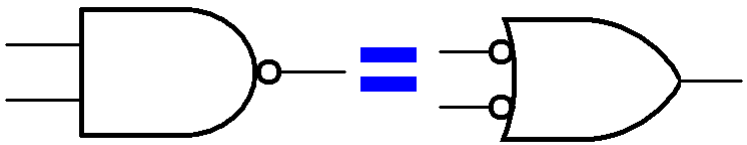
These are known as the **deMorgan theorems**.

The first theorem has four possible states, as shown on the next page. You see that in each case the right and left halves are equal.

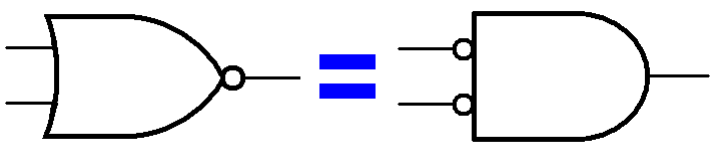
$\overline{A \bullet B}$	$\overline{A} + \overline{B}$
$\overline{0 \bullet 0} = \overline{0} = 1$	$\overline{0} + \overline{0} = 1 + 1 = 1$
$\overline{0 \bullet 1} = \overline{0} = 1$	$\overline{0} + \overline{1} = 1 + 0 = 1$
$\overline{1 \bullet 0} = \overline{0} = 1$	$\overline{1} + \overline{0} = 0 + 1 = 1$
$\overline{1 \bullet 1} = \overline{1} = 0$	$\overline{1} + \overline{1} = 0 + 0 = 0$

Yes, you need to go through each equation and verify that it's correct, for practice. And also because I'm going to make you develop the equations for the second theorem yourself.

From a gate standpoint, what the first theorem is saying is that a NAND is equivalent to an OR with inverters on its inputs. Trace out the gates; in either case both inputs have to go high for the output to go low.

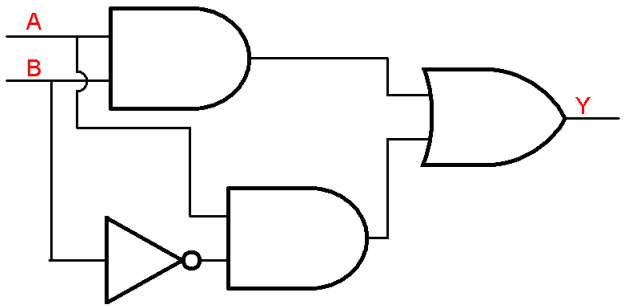


Another way of looking at it: **an active-high NAND is the same as an active-low NOR**. If either input or both inputs go low ("1" in active-low logic) the output goes high ("0" in active-low logic.)
The second theorem says a NOR is equivalent to an AND with inverters on its inputs:



And you probably guessed: **an active-high NOR is the same as an active-low NAND**. If both outputs go low, the output goes high.

SIMPLIFICATION



So: now we've got a lot of mathematical ammunition, let's see about that simplification stuff. Suppose you were designing a logic system, providing the appropriate gates for each function, and you wound up with something like this: Can this be simplified? It isn't obvious from the diagram, but it will become obvious if we develop the equations. The two inputs are ANDs, which are multiplications. Leaving out the dots, the expressions are

upper function: $A\overline{B}$

lower function: $A\overline{B}$

combined (ORed): $Y = AB + A\overline{B}$

Now, we can use the distributive rule:

$$AB + A\overline{B} = A(B+\overline{B})$$

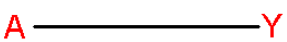
And we can apply the complementation rule:

$$B + \overline{B} = 1$$
$$A(B+\overline{B}) = A(1) = A$$

Which means, if we go back to our original equation:

$$Y = AB + A\overline{B}$$
$$Y = A$$

In other words, B is irrelevant: Y is simply A, and the whole 4-gate circuit is equivalent to a piece of wire.

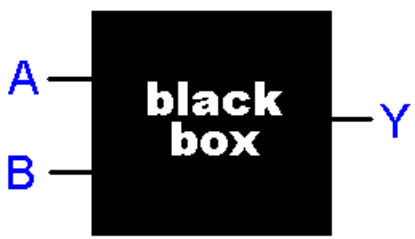


Wouldn't you feel stupid if you built the circuit, and then somebody pointed this out to you? But that's not going to happen, because now that you know how to simplify Boolean expressions, you're going to be the one making other techs feel stupid. Don't gloat, it's bad form.

KARNAUGH MAPS

It is important to understand that not every truth table corresponds to one of the classic Boolean gates. An engineer building a physical system, or a computer programmer writing a program, must be free to impose *any* relationship between the states of the inputs and the output. However many inputs there may be, the designer must always have the freedom to say, "*this* combination results in a 1, *that* combination results in a 0," with no constraint except that the result should be useful.

After all, we're often talking about switches and sensors in a machine, and the whole point is to make a machine work the way we want it to, which is not always the way any standard Boolean gate works. The machine has a truth table of its own. Once a machine's truth table has been designed, if it doesn't match a standard Boolean gate, then the function is built up of multiple Boolean gates, or their software equivalents.



A	B	Y
0	0	0
0	1	1
1	0	0
1	1	1

Consider this black box. Its function does not correspond to any standard Boolean gate.

Since its function doesn't correspond to any Boolean gate, we have to write separate expressions for each of the four possible conditions. Let's make our job a little easier. Let's say that “1” is “true” and “0” is “false,” and we're only interested in equations that result in Y=true. There are two of those:

$$\overline{A} \cdot B = Y$$
$$A \cdot B = Y$$

We OR those to get the complete picture:

$$\overline{A} \cdot B + A \cdot B = Y$$

In plain English, this says there are two input conditions, 01 and 11, that result in a true (logic 1) Y.

Now we simplify.

$$\overline{A} \cdot B + A \cdot B = Y$$
$$B(\overline{A} + A) = Y \qquad \text{regroup}$$
$$B(1) = Y \qquad \text{substitute 1}$$
$$B = Y \qquad \text{discard 1}$$

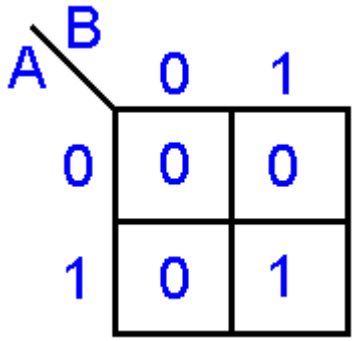
Look again at the truth table; Y = 1 when B = 1 and Y = 0 when B = 0. A is irrelevant. We're back to the situation on the previous page, where a bunch of gates can be replaced with a wire.

I've used a trivial example because I don't want you to have to wrestle with the math while you're learning about Karnaugh maps, since **the purpose of a Karnaugh map is to avoid the math entirely**. So now let's look at what a Karnaugh map is and how it would handle this same problem.

A Karnaugh (KAR-no) map or **K-map** is a spreadsheet, with the possible states for each input along the top and side, and the resulting output states in the squares. You read across and down, like a sports statistics graph.

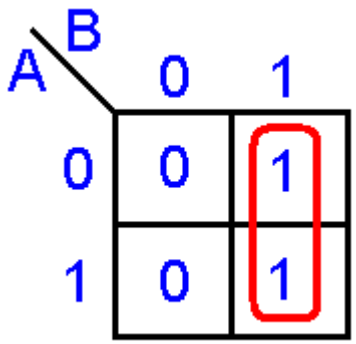
A simple, single gate has four possible input states – 00, 01, 10, 11 – and a specific output for each of the four combinations. The Karnaugh map has four boxes, four cells, to fit those four conditions.

For example, an AND is mapped as shown to the right. A and B are inputs; the Y output state for each combination of 1s and 0s is shown in the corresponding cell. Note that, reading across from A=1 and down from B=1, we see that Y=1, and that is the only square with a 1 in it.



In a Karnaugh map, what we want to do is find 1s in a group. The number of 1s must be a power of 2, which is to say, a group of 2, 4, or 8 of them. A group of three or six 1s is illegal. We can go horizontally or vertically, but not diagonally.

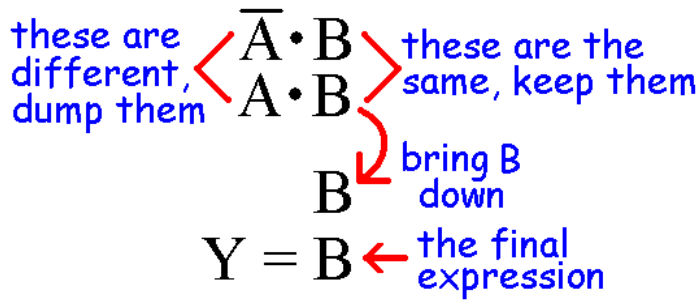
The 2-variable K-map for our black box is shown to the right. There is an obvious vertical grouping of two 1s. They represent the combinations notA·B and A·B respectively. We write those above each other.



$$\overline{A} \cdot B$$
$$A \cdot B$$

Now we "bring them down." The rules are:

- If terms above each other are the same, they stay. (We bring them down.)
- If terms above each other are different, they cancel. (We don't bring them down.)



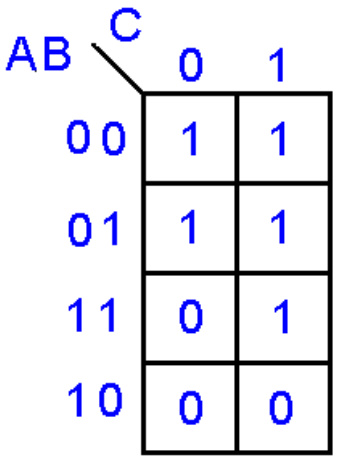
Even at this simple level, the K-map is probably easier than the math. When we go to three variables, the K-map is a *lot* easier than the math.

For a 3-variable example, let's use the truth table shown to the right. Once again, it does not correspond to any standard Boolean function. There are eight possible combinations of input states, meaning we have to write eight equations and then simplify them. If we use our trick of writing only the equations that result in Y = 1, we still have to write five equations.

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

But we're not going to do that. We're going to use a K-map.

The K-map is 2-dimensional, how do we handle 3 variables? Well we could do a 3-dimensional K-map, but we don't have to. We simply make a 2-by-4 K-map with eight cells. Then we take two of the four variables and write the possible states down the left side, as shown to the right.



There's something screwy, though. The numbers down the left side don't go in binary order. Is this a mistake? Nope. In order for the grouping of 1s to work properly, we have to arrange things so only one variable – only one bit – changes at a time. That means we can't list A and B in binary.

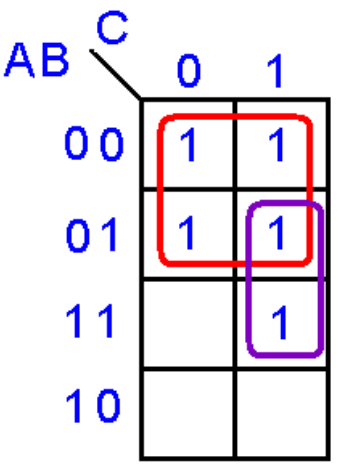
00
01 -one bit changes
10 -two bits change
11 -one bit changes

So we rearrange things:

00
01 -one bit changes
11 -one bit changes
10 -one bit changes

We're not breaking any rules, because we're not counting, we're just making a list. While we're at it, let's do a couple of other things. Since we're only interested in the cells where a 1 appears, let's stop bothering to put in 0s. That makes the groups of 1s more obvious. And we can stop bothering with the dot that signifies AND (multiplication).

That screwy number order is called **Gray code**, after a dude named Frank Gray who invented it. We're going to run across it again in chapter 8.



What we want to do is capture as many 1s as possible in as few groups as possible. That will result in the most simplified final expression. There are many ways the ones in this map could be grouped, but the two groups shown do it most efficiently. Note that it is “legal” to use a 1 more than once.

Now we write out the expressions represented by the 1s in the groups. There are five 1s, but since one of them is used twice, there are six expressions that are “true.”

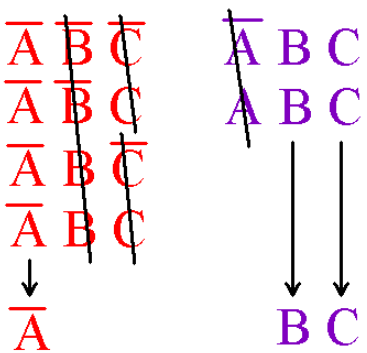
$\overline{A}\,\overline{B}\,\overline{C}$ $\overline{A}\,B\,C$

$\overline{A}\,\overline{B}\,C$ $A\,B\,C$

$\overline{A}\,B\,\overline{C}$

$\overline{A}\,B\,C$

Up to this point, we've only been doing the work that we would have had to do in any case. The next step is to simplify, which would be a chore if we did it mathematically. Instead, we cancel and bring down in Karnaugh fashion:



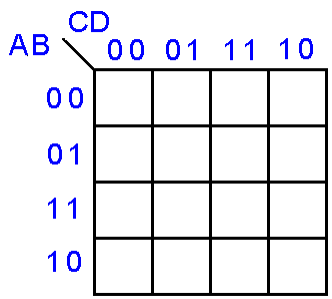
We now have two simplified expressions: Y is 1 any time A is false (that is, 0), or any time BC is true (that is, 1).
You can verify that this is correct by looking at the K-map. Observe that whenever A is 0, the result is always 1, no matter what B and C are doing. And whenever BC is 1, the result is always 1, no matter what A is doing.

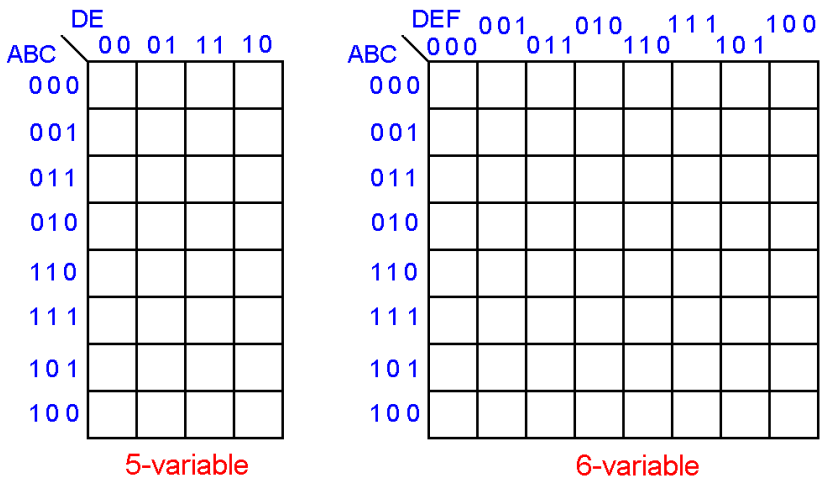
Y is 1 any time one **OR** the other “true” condition or both are present. That means we OR the expressions, as we did when we were using math to simplify, to get the final expression:

$$Y = \overline{A} + B\,C$$

If you want to say “Presto Changeo” when writing out the answer, go ahead. It's kind of magical. Especially to techs who don't know K-maps.

Ready for a 4-variable K-map? We put two variables on top and two down the left side. We have to follow Gray code on top as we did on the side. We have twice as many cells, which shouldn't be a surprise: every added bit doubles the number of possible combinations. Five and six variables would double the size and double it again (next page):

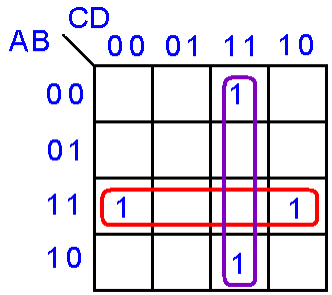




Take a moment to study the Gray code. In every case, only one bit flips or flops between numbers. This is true of Gray code no matter how many bits there are.

KARNAUGH RULE SUMMARY

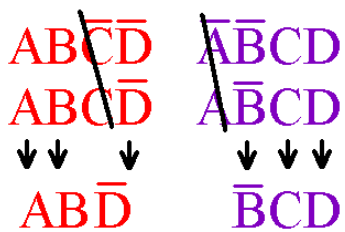
- Find groups of 1s. The goal is to capture every 1 with as few groups as possible.
- Groups must contain 2, 4, 8, 16... 1s, in other words, a power-of-2 number.
- Groups cross cell boundaries vertically and/or horizontally, but not diagonally.
- Any 1 can be a member of more than one group.
- 1s must be adjacent, but it is legal to "go around the ends" of the map, counting 1s on the edges as adjacent.



- Once the groups are identified, the expressions for the variables that created those groups are written out. For the map at right, they would be:

$AB\overline{C}\overline{D}$ $\overline{A}\overline{B}CD$
$ABC\overline{D}$ $A\overline{B}CD$

- If the terms in a column are different, they are discarded. If they are the same, they are retained.



- Compare the reduced expressions to the K-map to verify that you didn't screw up somewhere.

- OR the remaining expressions to find the reduced, boiled down, mystic, at-one-with-the-universe, final expression.

$$Y = AB\overline{D} + \overline{B}CD$$

I know what you're thinking: “Did I need to learn this?” Frankly, I don't know. Beyond about six variables, Karnaugh maps get clumsy, though a function with many variables can usually be broken into several functions with few variables. Program development software is used a lot more often than not.

Still, though there are two calculator apps on my smartphone, I do simple arithmetic in my head all the time. The more you outsource your thinking to machines, the more dependent you are on them. When a machine-dependent person hasn't got access to the machine, they're functionally a moron. I've talked to a number of people who use expensive software at work but K-maps for their own projects.

You're gathering tools at this point. In chapter 2, you picked up binary concepts, in this chapter you learned about binary logic, and in the next chapter you'll learn about sequential, time-based logic. Then, in chapter 5, we will look at microcontrollers, which use all the techniques you've learned. Microcontrollers are everywhere these days, and Boolean operations are a basic part of what they do.

If you haven't fully mastered the material in this chapter yet, that doesn't mean you're dense. Learning some of this stuff is like swallowing a bowling ball. If you've absorbed most of it, I would suggest that you move on to the next chapters – they're all easier than this one – and come back here now and then to review, until you've got the rest of it locked in.

End of chapter 3

EM4 Chapter 4
SEQUENTIAL LOGIC

Objectives of this chapter:

- *explain the difference between Boolean and sequential logic*
- *discuss "synchronous" and "asynchronous"*
- *explain the function of the clock signal and the requirements for the clock generator*
- *define set, flip, reset, flop, clear*
- *explain positive-going and negative-going clocks*
- *discuss serial and parallel data, and the circuits used to perform conversions*
- *explain ripple counters and shift registers*
- *discuss gated and SR latches*
- *discuss T, SR, D, JK flip flops*
- *discuss BCD in the context of frequency counters*

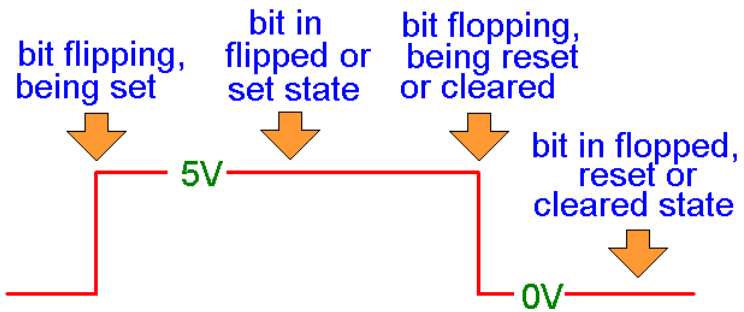
SEQUENTIAL LOGIC

The chapter 3 discussion of Boolean logic did not include more than a passing mention of *time*. The output state at any moment is simply the result of the input states at that moment, within a few nanoseconds. In **sequential logic**, on the other hand, the circuit is in state "A" until "X" happens, and then it's in state "B." In other words, Boolean logic is "now" while sequential logic has a definite "before" and "after." To deal with this additional factor, we need additional concepts and words. I'm going to list and define them now, so we'll have them when we get to the circuits that follow.

Words: Set, Reset, Flip, Flop, Clear

With many sequential logic circuits, when a bit enters a 0 or 1 state, it's latched in that state, like a snap-type light switch. When it's time to change state, the circuit instructs it to change state, and then it's latched in the other state until it's told to change back. There are several words used to describe this:

Flip: when a bit changes from 0 to 1. Think of standing a domino on its end.
Flipped: a bit in the 1 state.
Flop: when a bit changes from 1 to 0. Think of knocking the domino over.
Flopped: a bit on the 0 state.
Set: to flip, or be flipped.
Reset: to flop, or be flopped.
Clear: same as flop.

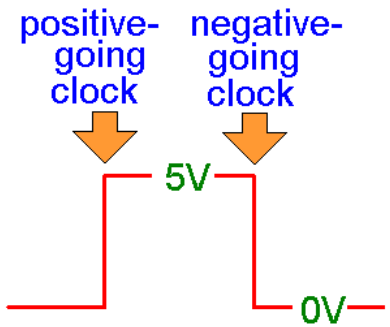

bit flipping, being set
bit in flipped or set state
bit flopping, being reset or cleared
bit in flopped, reset or cleared state
5V
0V

Words: Synchronous, Asynchronous, Clock

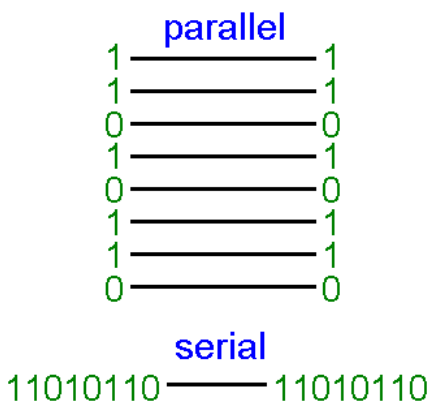
Asynchronous: an event that happens as soon as the necessary conditions are in place, also refers to a circuit that operates asynchronously. In Boolean gates, when the input states change, the output states change almost immediately. They are **asynchronized**, without synchronization to an external time reference.

Synchronous: an event that happens in response to a timing signal, also refers to a circuit that operates synchronously. In some cases, the function is the same as a Boolean operation, but the circuit waits for "permission" from a timing signal before it acts. You can change the input states all day long and nothing will happen until the gate receives a pulse at another input that says it's time to act. In this way the operation of the circuit is **synchronized**, made to operate in lock-step like people marching to a drummer.

Clock: the timing signal that makes a synchronous event happen. This can be a one-time pulse or, more often, a continuing series of pulses that occur at regular intervals. Clocking may occur at the instant when the clock transitions from 0 to 1, which is called a **positive-going** clock, or from 1 to 0, which is called a **negative-going** clock.



Words: Serial and Parallel



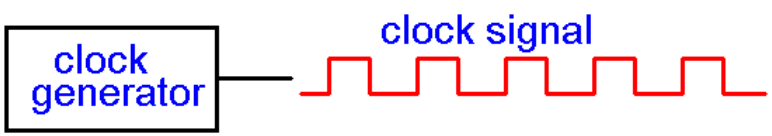
There are two ways of transferring a binary word from one place to another.
Parallel: a word is transferred all at once, over parallel wires. This theoretically has the advantage of speed. **Serial**: a word is transferred one bit at a time, over one wire. This has the advantage of low equipment cost.

We're ready (that is, we're set, but hopefully not flipped) to deal with circuits that go beyond Boole. Okay, that was lame even for an old fart. But you don't see me deleting it, do you? People say I have no pride, and they're right.

In the discussions that follow, assume we're always dealing with active-high circuitry unless otherwise stated.

THE CLOCK GENERATOR

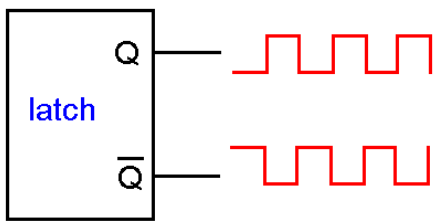
For circuits that use a clock, the clock should be sharply defined; that is, it should be a square or rectangle wave with fast rise and fall times and flat tops and bottoms. We must either use a circuit that generates this kind of waveform, or take less-defined waves and use a Schmitt buffer to square them off.



If you've done the EM3 study guide you know that, broadly speaking, the word **oscillator** means a waveform-generating circuit whose frequency is determined by a tuned circuit, and **multivibrator** means one whose frequency is determined by RC time constants. Multivibrators generate square waves naturally, but many digital applications require a precisely controlled clock frequency, which can only be obtained via a tuned crystal oscillator or one of its new-technology equivalents. Tuned oscillators usually generate sine waves. In these cases, the oscillator waveform is squared off by routing it through a **differentiating** circuit, also covered in EM3, which provides a precise, sharp-edged 0-1 or 1-0 transition.

LATCHES

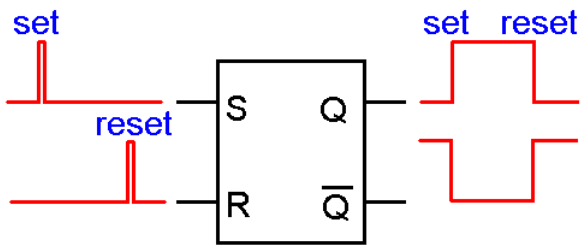
A latch is a gate that catches an input state, transfers it to its output, and keeps it in that state even after the input has changed. The output is labeled **Q**, and there's usually a second output labeled **NOT-Q**, often abbreviated nQ in text, and indicated by a line over the Q in the schematic symbol. The nQ output is the same as the Q output, but inverted: when Q is 1, nQ is 0 and vice-versa.



There are a number of kinds of latches, and some other gates that incorporate latch functions. We'll start with asynchronous latches, which latch immediately when the proper input conditions occur.

THE SR LATCH

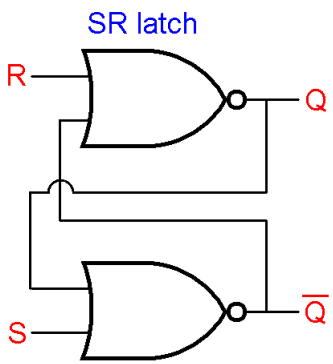
The most basic is the **SR latch**, which has two inputs, **set** and **reset**.



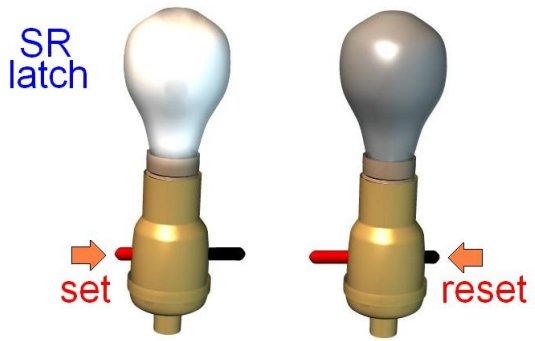
In a SR latch, “1” on the S (set) input will flip the Q output. “1” on the R (reset) input will flop the Q output.

This is also known as an **RS** latch – just some people's preference – and an **SC** latch for **set** and **clear**. Set and reset happen immediately, so the SR latch is asynchronous.

In its simplest form, the SR latch is made by cross-connecting a pair of NOR gates. Let's assume the gate is initially reset, meaning the Q output is low and the nQ output is high. Now we apply a 1 to the lower input of the lower gate. Its output goes low. This puts an 0 on one of the upper NOR's inputs, and since

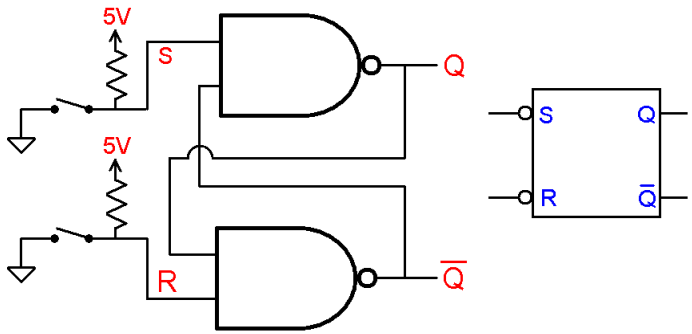


the other one is already low, its output goes high. This puts a 1 on one of the lower NOR's inputs, holding its output low even after the other
input returns to 0. Thus it is latched in the flipped state; a 1 on the input to the upper NOR will reset the circuit.



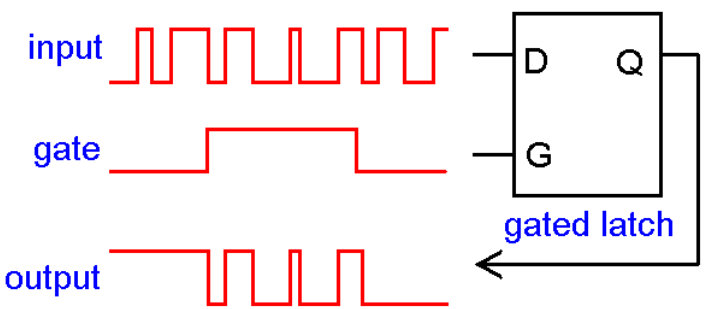
An SR latch is like a lamp socket with buttons sticking out either side. One turns the lamp on, the other turns it off.

In many circuits, a latch input will go low to activate a function instead of high – in other words, the latch inputs must be active-low. An active-low RS latch can be made by using NANDs instead of NORs.



THE GATED LATCH

In a gated latch, as long as the gate input is 1, the output will follow the input. When the gate goes to 0, the output freezes in whatever state it's in at that moment, and stays that way no matter what the input does.

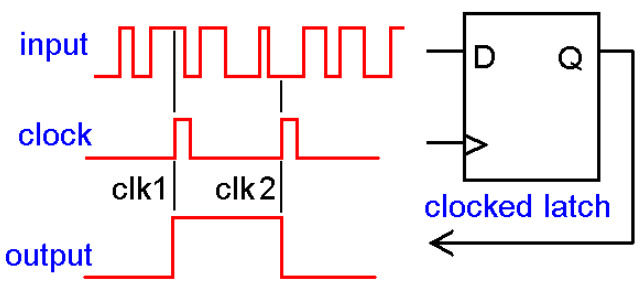


In the last chapter, we studied an enable input that disables the output by putting it into a high-impedance state. This is a different kind of enable; when it goes low, instead of the output going to Z, it stops following the input and freezes.

The input is labeled D for Data, the gate is labeled G, and the output is Q; the latch may also have an nQ output.

THE CLOCKED LATCH

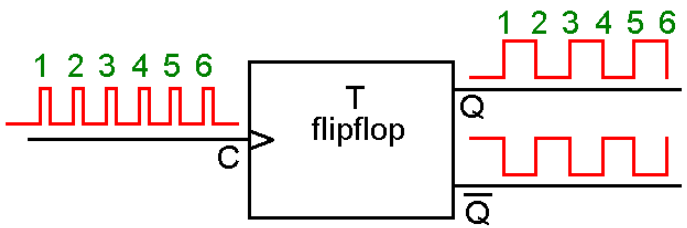
The clocked latch is our first synchronous circuit. In a clocked latch, the input state is transferred to the output (**clocked**) at the moment when the clock input goes from 0 to 1. For the output to change state again, the clock must go to 0 and then to 1 again. This is called an **edge-triggered** function. In a schematic diagram, an edge-triggered clock input is indicated by a little triangle as in the diagram above right.



A D-latch often takes the form of a standard SR flipflop and an inverter. Trace out imaginary highs and lows until you see how it operates.

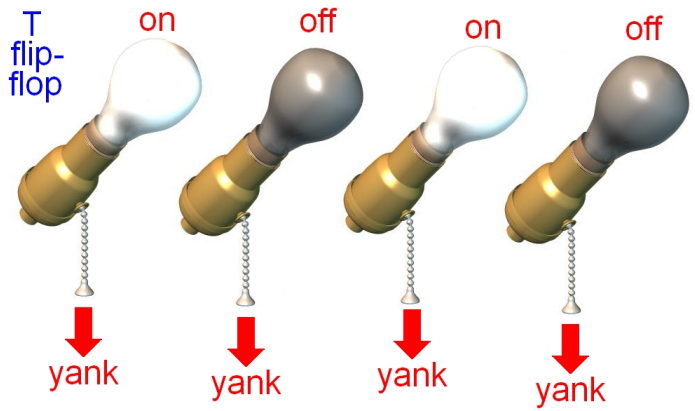
THE "T" FLIP-FLOP

The **flip flop** or **FF** is the basic building block of binary

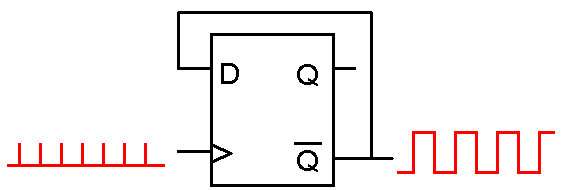


counting circuits. In its classic form, there is an edge-triggered clock input and an output. The clock input is labeled **C**; the outputs are labeled Q and NOT-Q. On the first clock, the Q output goes from 0 to 1. On the second clock, it goes from 1 to 0. On the third it goes from 0 to 1 again. In other words, the output alternates in response to clock inputs, 1-0-1-0-1-0.

The name "flip flop" describes the alternating operation; on each clock, the output alternately flips and flops. This is also called **toggling**, which is why this kind of FF is often called a **T** flip flop. It is, obviously, synchronous, as it changes state in response to the clock. Think of a lamp socket with a pull chain. Repeatedly yanking the chain turns the light on, off, on, off. That's a basic T flip flop.



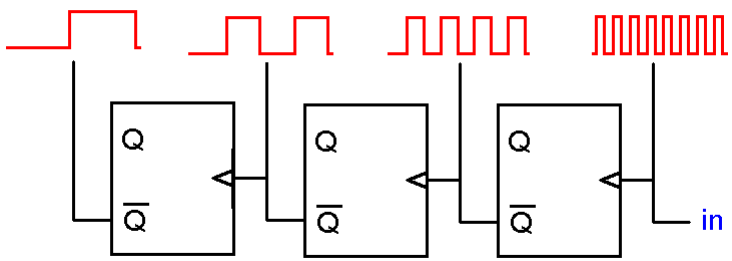
A T-flipflop is a divide-by-two circuit. The outputs go through a flip/flop cycle once for every two input flips.



A manufacturer often makes a T-flipflop – and you can make one too, when you haven't got a T-flipflop chip handy – by taking a clocked latch and attaching its nQ output to the D input. I'm going to make you figure out how it works because doing so will help you understand the more complicated stuff that comes later.

THE BINARY COUNTER

If we connect the nQ output of a T-FF to another T-FF's clock input, we'll have a circuit where, when the first FF's Q output flops, the second flips. That's the main use of the nQ output.
Let's connect three FFs that way, and turn the circuit around so the input is at the right and the chain of FF's goes to the left. If we toggle the input, we will see that each successive output frequency is half that of the previous output. The output sequence is as follows: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000.
We've just created one of the most basic building blocks of a computer, a circuit that counts in binary, called a **ripple counter**.

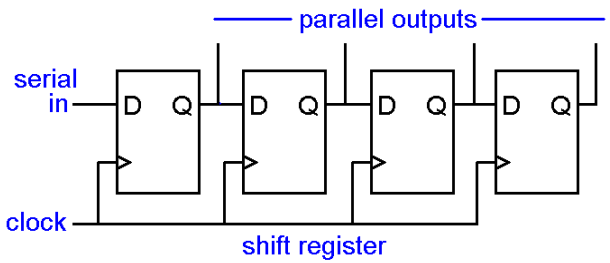


THE "D" FLIP-FLOP

The clocked latch we studied has another name: **D** flipflop, for **delay**. The input state appears at the Q output when the clock goes positive; the gate delays data from input to output by one clock. In other words, it doesn't toggle but rather shifts the input state to its output when the clock occurs.

THE SHIFT REGISTER

If we take a bunch of D flipflops, connect Q outputs to D inputs, and clock them all together, we'll get a circuit that transfers the input state down the line of latches, one gate per clock. This is called a **shift register**.



At a dinner party, Emily Post would say food should be passed from left to right around the table. You pick up the salad, take some, pass it to the person on your right. They take some and pass it to the person on their right, while you pick up the bread, take a slice, and pass it to the right. The pickles might come next. You wind up with a bunch of food being passed around at the same time, in bucket-brigade fashion. That's a shift register.

"Shift register" is often abbreviated **SR**, which could be confused with an SR flipflop. Usually, the context will make clear what "SR" means.

A SR can have just a clock input, a data input, and a data output. More commonly, an output is tapped off between each pair of FFs as shown to the left, so a 4-bit SR will have four outputs. If you connected 4 LEDs (with series resistors to control current, naturally) you could watch the bits move along the SR string. You could even add power transistors and make a sign with Las Vegas style chase lights.

SERIAL IN, PARALLEL OUT

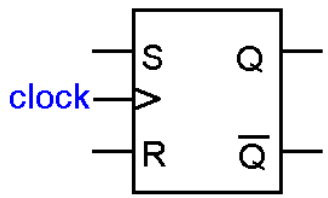
A shift register with outputs between each pair of FFs is a **serial-in, parallel-out** shift register. Data goes in as a serial bit stream, one bit wide. It comes out as a serial bit stream, or as a parallel word of however many gates there are in the circuit. One application is to turn a serial data stream into parallel bytes.

PARALLEL IN, SERIAL OUT

Sometimes we want to go the other way, loading a whole word into a SR, then clocking it out one bit at a time, converting parallel data to serial. In these cases we would use a **parallel-in, serial-out shift** register, which loads a binary word all at once and then shifts it out bit by bit.

THE SR FLIP FLOP

An SR FF is the hybrid offspring of a T-FF and an SR latch. As long as the clock is low, the S and R inputs can toggle all day and the outputs won't change. However, when the clock goes high, the FF flips (or stays flipped) if the S input is high, and flops (or stays flopped) if the R input is high. The system should be designed so S and R are never high at the same time; that is an illegal condition.



The illustration shows an edge-triggered clock input, where Q and nQ flip or flop, depending on the states of S and R, on the rising clock edge. There is also a non-edge triggered SR flipflop, where the clock is a simple gate. As long as the clock is high, the S and R inputs can change state and immediately affect the Q and nQ outputs. When it is low, Q and nQ do not change, just like the edge-triggered version.

An SR-FF is also called an **SC-**FF, for **set / clear**.

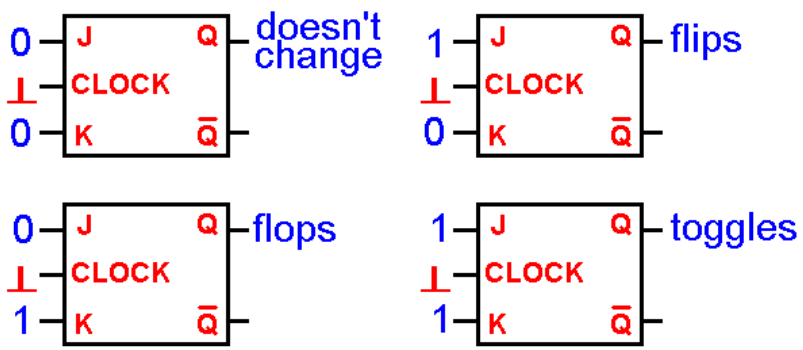
THE JK FLIP FLOP

In most ways, the **JK** is like the SR FF. The first three states are the same:

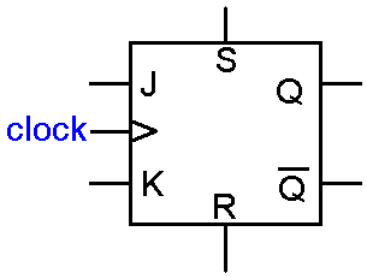
- J and K both low: the output does not toggle; that is, operation is frozen.
- J high, K low: the Q output will flip on the next clock, or remain high if it is already flipped.
- K high, J low: the Q output will flop on the next clock, or remain low if it is already flopped.

But what happens when both J and K are high is different. In the SR latch, this is a forbidden condition – the circuit designer is supposed to make sure S and R are not both high at the same time. But in a JK-FF, that's a fourth "legal" state:

- J and K both high: the output toggles on each clock like a T FF.

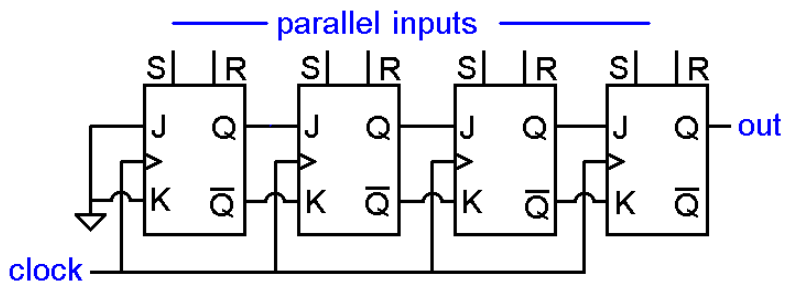


I don't know where J and K came from, but I think of "jack" and "kill" to keep them straight. I'm sure somebody will email me saying "No, you jerk, this is what they mean..."



Most JK FFs also have S and R (or C) inputs, so you can choose whether to cause a flip or flop action when the next clock occurs, or force it immediately. This fully loaded JK FF is a "universal" FF; you can make it act like any other kind using the S, R, J and K inputs.

The parallel-in, serial-out shift register mentioned on the previous page can be built with JK shift registers. Each FF's J and K inputs are connected to the previous one's Q and nQ outputs, so data is moved serially along and out of the string. S and R inputs can be used to jam parallel data into the Ffs.

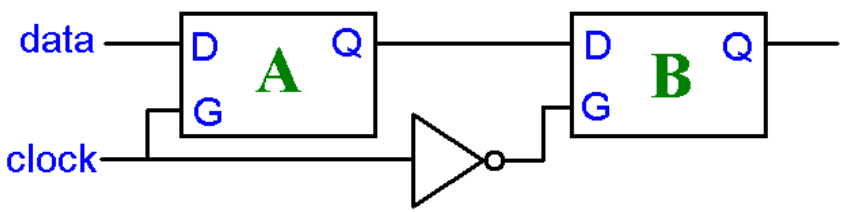


When used this way, S may be labeled **PR** or **PS** for **preset** and R labeled **CLR** for **clear**.

MASTER/SLAVE

A constant risk with digital circuitry is that extra pulses will get through, due to sloppy gating or some other cause, and screw up the count or sequence. One way to increase reliability s to use two gates that operate one after the other.

In the diagram below, when the gate input goes high, latch "A" transfers data from its input to its Q output. When the gate goes low, the inverter output goes high, sending a high gate to latch "B," which transfers latch A's output to its own output. This way, we get a D flipflop with true, reliable edge-triggering. The gate must go all the way up and all the way down for data to pass through. There is no chance for weird operation or for data to shoot through from input straight to output. Think of a spaceship with an airlock, where one door must close before the other one opens.

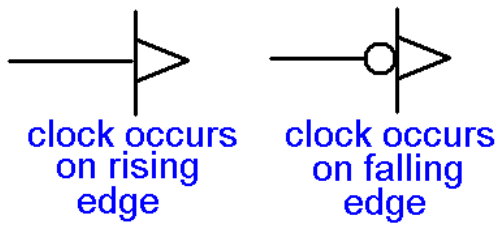


This is called a **master / slave** arrangement; latch "A" is the master and latch "B" is the slave. Many JK flipflops are built this way. It's probably the best form of edge-triggering.

ACTIVE-LOW TRIGGERING

Not all edge triggers are positive-going triggers. In the master / slave example above, the data appears at the output of latch B when the input clock *falls*. This is a negative-going trigger, or it could be considered an active-low trigger.

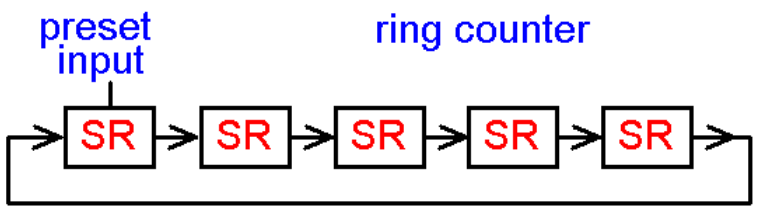
Falling-edge triggers are fairly common, indicated by the usual inversion circle. One reason for using a falling-edge trigger is that a ripple counter can be built using T-FFs with only Q outputs, which is a simpler circuit. I'll leave it to you to figure out the schematic.



I'm going to close this chapter with a couple of other types of circuit, which use both Boolean and sequential logic.

THE RING COUNTER

The ring counter is a shift register whose output is connected back to its input, so the bit pattern recirculates. When the circuit is first powered up, it's necessary to load a 1 into the first FF. After that, the 1 is shuttled around the ring endlessly.



UP-DOWN COUNTER

The usefulness of a binary counter is greatly increased if it can count down as well as up. This involves such a complex circuit I'm not even going to show it to you. Since we're talking about an IC chip, you don't have to care what goes on inside. One of the leads can be set or reset to make the counter go up or down; other leads handle "carry" and "borrow" operations so counters can be strung together to handle bigger numbers.

Presettable up/down counters allow you to jam a number in to preset the starting count. Parallel in, right. One reason you'd want to do this is that it's a lot easier to make a circuit that tells you when it has counted *down* to zero than tell you when it has counted *up* to a specific number. Suppose you want a machine to count out $3C widgets. (That's $3C_{16}$ or 60_{10}.) You'd preset the counter with the number $3C and then the machine would count down and stop when it hit zero. Microcontrollers (next chapter) have that built in.

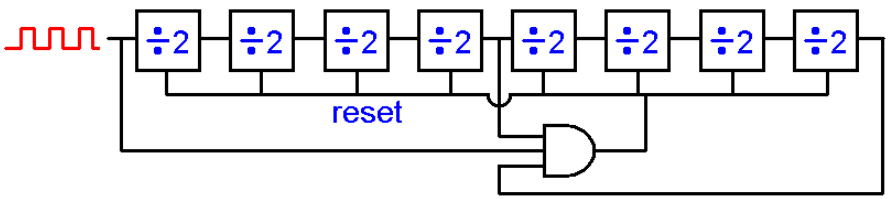
MODULUS COUNTERS

In many cases, we want to count a number that is not a power of two. I've mentioned that an oscillator that must generate a precise frequency usually uses a quartz crystal. If we're designing a radio that has to work on a number of different frequencies, and the frequencies have to be accurate – as is usually the case – that means we need a number of different-frequency crystals. Or, we can have one crystal at one frequency and use it as a reference to generate other frequencies. This is one form of **frequency synthesis**.

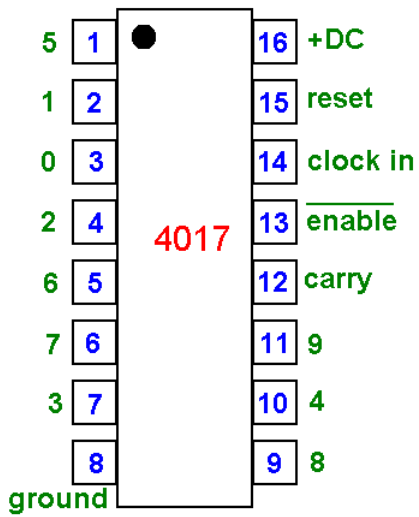
Suppose we choose a 1MHz crystal, and on a particular day we want to tune the radio to 72MHz. What happens inside the radio is that a variable-frequency oscillator makes something close to 72MHz. A divide-by-72 circuit turns this into about 1MHz, another circuit compares this to the precise 1MHz coming from the crystal oscillator, and a third circuit uses the error voltage, if any, to adjust the 72MHz oscillator until it's creating exactly the right frequency.

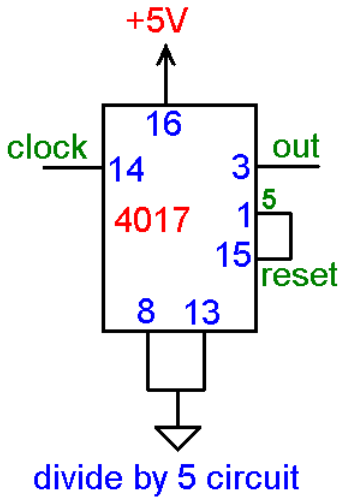
That's one use for a phase-locked loop, which we studied in EM3.

Mathematicians like to use words that make them look even smarter than they are. They would say that a divide-by-72 counter has a **modulus** of 72. It's made by putting together enough binary circuitry to count up to 72, then on the 73rd count, reset to zero. One way of doing this would be to create an 8-stage ripple counter, but instead of letting it count all the way to 255_{10}, use an AND to trip the reset when it gets to 73_{10} (1001001_2).



The short name for modulus counter, which everybody uses, is **mod counter**. Probably the most used example is the divide-by-10 or mod 10 counter. For example, the old standby 4017, introduced about 1968, counts to 10 and has 10 individual outputs. For every ten input pulses, every one of these outputs flips, one after another, and flops again on the next count. When the last one flops, returning the counter to zero, a “carry” output goes high. This permits chaining several 4017s together.





If the reset pin is connected to ground, the 4017 will just count in 10s. However, if one of the 10 outputs is connected to the reset pin, the count will be reset before it reaches 10. For example, if you connect the #5 output (pin 1) to the reset, the chip will count up to 5 and then reset. In other words, the modulus is adjustable.

BCD COUNTERS

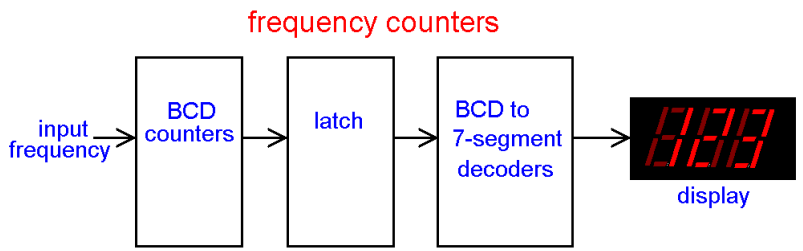
Another count-to-10, mod 10 chip is the **binary coded decimal** or **BCD counter**. Unlike a 4017, with 10 outputs, a BCD counter's output is binary. It's simply a 4-bit ripple counter that is not allowed to count all the way up to 1111_2. It's programmed to reset as soon as the count goes past 1001_2, which is 9_{10}. That makes it a kind of diplomat that bridges the gap between the way computers think and the way people think.

BCD-TO-7-SEGMENT CONVERTER

Since we're talking about BCD, I should mention the BCD converter chip, which is an IC that takes the BCD word and converts it to an output designed to drive a 7-segment number display.

A 7-segment display consists of 7 rectangle shapes that can be illuminated. They're physically arranged so any decimal number can be produced by turning on the right lights. Usually, the lights are LEDs with either their anodes or cathodes connected together and the other 7 terminals available to be connected to a BCD-to-decimal decoder chip. There's also an eighth LED to illuminate a dot, so we can display a decimal point.

As an example of an application for these circuits, a frequency counter combines BCD counters, which count cycles, a latch circuit, which periodically transfers the BCD count to the decoders, and decoders which translate BCD to 7-segment displays.



There was a time when computers were built up of all these different kinds of chips. I used to have one; the central processing circuit was an 18-inch square PC board, and if you were masochistic, you could look at the individual chip part numbers and trace out the circuit.

BCD counter and decimal converter chips, and for that matter many single-function digital chips, are used a lot less today than they once were. But you haven't wasted your time learning about them, because the functions still constitute the foundation of digital technology. They're just built into microprocessor chips today. So, now it's time to talk about programs and microprocessors. Right this way madame, mademioselle, monsieur, chapter 5 awaits.

End of chapter 4

EM4 Chapter 5
MICROCONTROLLERS AND PROGRAMMING

Objectives of this chapter:

- *describe the parts, functions and interconnections of the components that make up a microcontroller.*
- *discuss and compare hardware, software and firmware.*
- *describe ROM and RAM, and define "volatility."*
- *describe PROM, EPROM, EEPROM*
- *define "instruction set" and "RISC" and give examples of common instructions*
- *describe how a flow chart is constructed*
- *explain the function of the clock signal*
- *explain the function of timers, including the watchdog.*
- *explain the differences between machine, assembly and high-level languages*
- *explain the types and functions of interrupts*
- *describe how an analog to digital converter can be made using a microcontroller's timing and comparator functions*
- *explain the function of serial-to-parallel and parallel-to-serial converters*
- *describe the UART and the function of its buffer*

In a study guide this size, I can only give a basic, skim-the surface introduction to computers. The physical part of a computer system is called **hardware** – computer, monitor, keyboard, printer etc. The non-physical part, which consists of the binary instructions that operate the computer, are called **software**. If those instructions are installed once and never changed, so they're about as permanent as the hardware, they're called **firmware**.

MICROCONTROLLERS

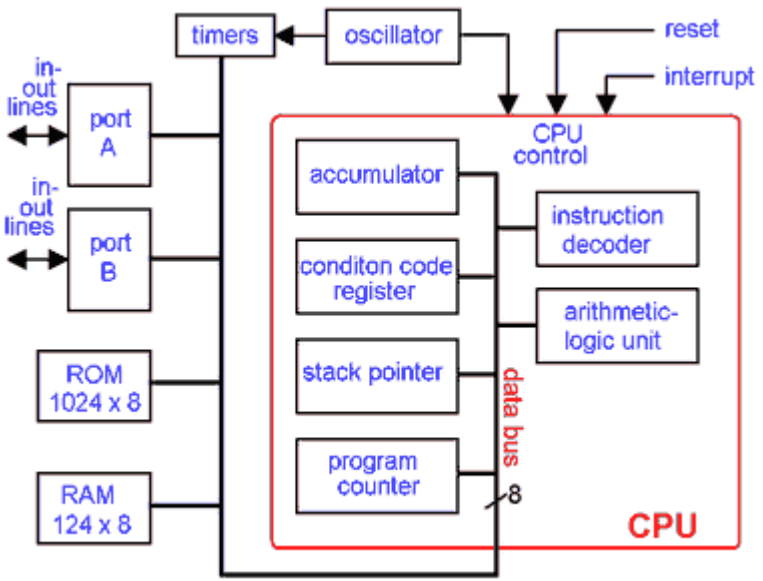
Instead of starting with full-fledged computers, in this chapter I'm going to talk about the **microcontroller**, which is a complete small computer on a single chip. First, microcontrollers are less complicated. Second, as a tech, you're not going to go inside a computer very often unless you become a computer specialist. But you're likely to deal with microcontrollers all the time. They're everywhere.

I was at a laundromat today, where every washing machine and every dryer has a microcontroller – and these machines are twenty years old. I have an iron with a microcontroller in it, and an oven, and three clocks. My car has dozens. I just sent off a get-well card that plays a tune when you open it. That's a microcontroller, and it has more computing power than the whole world had in 1950.

The abbreviation for microcontroller is **MCU**, for **microcontroller unit** or, alternately, **microcomputer unit**. When a microcontroller is built into another device, like a washing machine, it's called an **embedded** microcontroller.

An MCU is a self-contained computer on a chip, with all the necessary circuitry to store data and instructions, run a program, and communicate with the outside world. A basic microcontroller is made up of the following components:

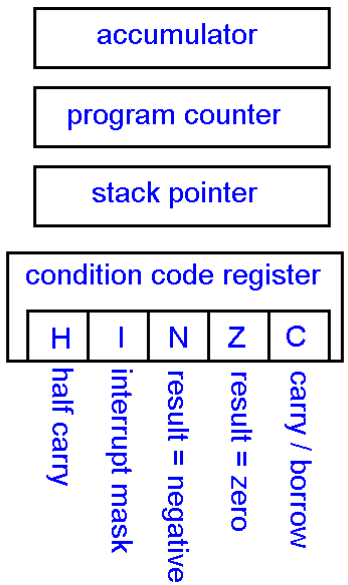
- Computation circuitry called the **arithmetic-logic unit** or **ALU**, which does all the computation and also handles data routing and management functions in real time. The ALU is the MCU's brains.
- Data storage, called **memory**, which holds the data that the microcontroller works with, and also the instructions which tell it what to do with the data. The memory in a microcontroller comes in two flavors, **RAM** or **random access memory**, where data can be stored and retrieved at will, like a file cabinet, and **ROM** or **read only memory**, that holds firmware.
- Inputs and outputs, called **ports**, which enable the microprocessor to interface with the outside world.
- Support and supervisory circuitry such as oscillators, timers and malfunction detectors.
- A **bus**, or several buses (not busses), like a highway that connects all the parts, permitting data to be moved to and from the ports and among the various sections of the microcontroller. In some MCUs, this bus also transports instructions; in others, there's a separate bus for instructions.
- An **instruction decoder**, which reads the binary command words in the program and turns them into actual operations.



The **CPU** or **central processing unit** is where the number crunching takes place. It consists of the ALU and the support circuitry that it must have immediate access to. The block diagram above shows a very simple MCU.

In addition to the main RAM and ROM, an MCU has several small, dedicated chunks of memory right in the CPU that serve the following functions:

- **accumulator**: a "scratch pad" that stores the data the ALU is working with at the moment.
- **program counter**: the MCU follows a set of instructions contained sequential memory locations. The program counter keeps track of the location of the next instruction.
- **stack pointer**: a typical program often takes a detour from its normal sequential path. The stack pointer keeps track of where the program was when the detour occurred so it can come back and resume at that point.
- **condition code register**: sets or clears bits according to the results of calculations. These can then be used to determine what happens next. The specific ones in the illustration above are:



in

  - carry / borrow: set to 1 if the last addition's sum overflows the accumulator, or the last subtraction's result is negative, requiring a borrow.
  - result = 0: set if the result of the last calculation is zero. This is often used to end a process after a certain number of repetitions. For example, if a short delay is required, a number is loaded into the accumulator, and then decremented until it reaches zero.
  - result = negative: set if the result of the last calculation is negative.
  - Interrupt mask: set to block interruptions when the ALU is working on a problem that must not be disrupted.
  - half carry: associated with BCD math.

OPERATING INSTRUCTIONS

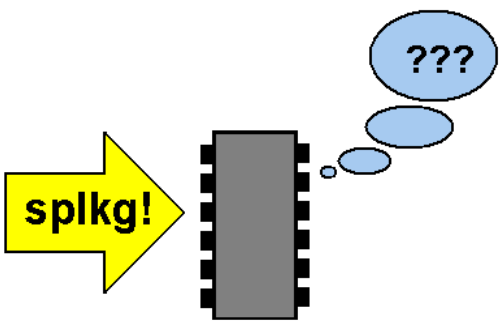
MCUs execute **programs**, which are sets of binary words that, instead of representing numbers, represent **instructions**, also called **commands**, that tell an MCU what to do.

The essential difference between a computer and an MCU, beside size, is that computer programs are mostly software while MCU programs are firmware.

Computer programs can be installed, changed and removed. MCU programs are usually permanent.

In most cases, an MCU is born to do just one thing all its life. When a new MCU is about to be put into service, a program is loaded into its ROM memory. From then on, it's an old dog that can't learn new tricks. It executes that program, and only that program, until that function is no longer needed. Then it goes into the dumpster.

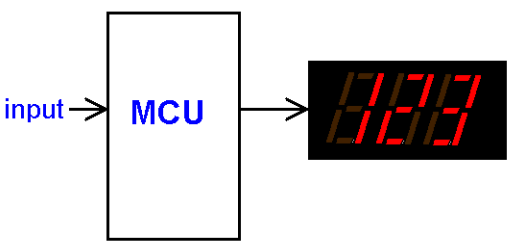
In EM3 we discussed the **ASIC** or **application-specific IC**, an analog chip custom-manufactured to perform a specific analog function. **Programming** an MCU essentially turns it into a digital ASIC, a special-purpose chip that may be the only one of its kind in the world. Yet another of digital's virtues is that special-purpose chips can be created easily, in small quantities, by programming general-purpose chips.



When a program is being developed, the most basic practical limitation is that it cannot contain any instruction that the MCU doesn't know how to execute. The list of operations an MCU can perform is called the **instruction set**, and it's part of an MCU's design, built in at the factory – the user cannot change it. If the MCU is going to be able to execute an instruction that says "move *this* piece of data from *here* to *there*," then the appropriate gates and data bus connections must be built into the MCU when it's manufactured. In other words, the instruction set is **hardwired**; it's embodied in Boolean gates and the like. It's up to the user to choose an MCU that offers the necessary instruction set.

It's possible to perform any digital function by hard-wiring a system together out of individual gate, flipflop, ripple counter, shift register, memory and other types of chips, and that's what people used to do. An MCU provides all these functions in one package, and the user supplies a program that tells the MCU how to use them. This usually means some of its capabilities are wasted – but with MCUs selling for a buck each in quantity, it hardly matters. This is the main reason many gate logic chips have become obsolete.

For example, in the last chapter I showed you a block diagram for a frequency counter made up of individual chips. I have an old one on my bench that's built that way, and it works well, but if I were building one today I'd just program everything into a single MCU.



Some MCUs feature a large instruction set for the sake of flexibility. Others only offer a short list, perhaps 20% of possible instructions that will handle 90% of applications. An MCU with a deliberately limited library of instructions is called a **reduced instruction set controller** or **RISC**. It's popular despite its limitations because of its simplicity and low cost and because the person writing programs for it doesn't have to study for years to memorize a large instruction set. When a complex operation is needed that the MCU does not support, a program can usually be written that performs the operation a different way.

A selection of the most common instructions:

- Arithmetic functions: counting up and down, addition, subtraction, multiplication, division. Some MCUs do not offer multiplication and division, which means these functions have to be performed by repeated addition and subtraction. This would be an example of “a program can usually be written that performs the operation in a different way.”
- Boolean logic functions: AND, OR, XOR etc.
- Standard gate, latch, and flipflop functions.
- Bit test functions: if a particular bit is 1, do *this*, if it's 0, do *that*.
- Bit set and reset functions.
- Binary word manipulation functions, e.g. shift all the bits in the accumulator right one position.
- Fetching data from memory, called **reading**, and storing data in memory, called **writing**.

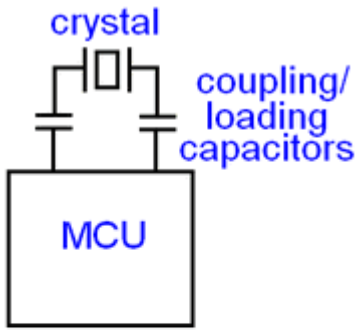
These instructions, the actual computing functions, are all handled by the ALU. Other parts of the MCU only execute support functions like keeping track of addresses and decoding instructions, like a corporate president's support staff. All decision-making goes straight to the Executive Office.

SEQUENTIAL OPERATION

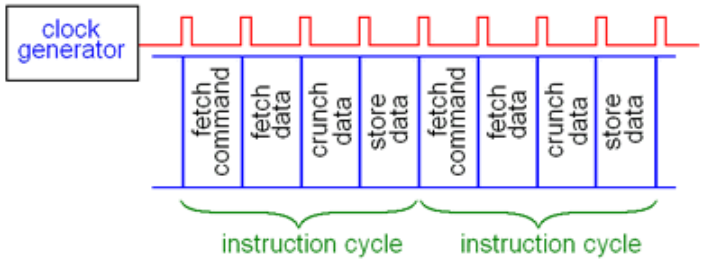
An MCU is a precision drill team, marching to a drummer, performing complex maneuvers in rhythm, one step at a time. With few exceptions, everything an MCU does is sequential. It pulls an instruction from its instruction memory, executes the instruction, then pulls the next instruction from memory and executes it, in a predetermined sequence. When something out of the ordinary happens, the MCU pulls up a set of instructions for dealing with the out-of-ordinary and follows those instructions sequentially.

The “drummer” is an oscillator that generates a continuing square or rectangular wave signal called the **system clock.** A crystal is used if a precise operating frequency is required as in the diabetes tester I use. Otherwise, the frequency is set by an RC time constant.

The clock frequency may be as low as 32kHz, especially if low power consumption is important, or as high as hundreds of MHz. Few microcontrollers run at GHz frequencies as desktop and laptop computers do. Almost every operation is tied to this clock, which is to say almost all operations are synchronous. Even Boolean logic is done synchronously.



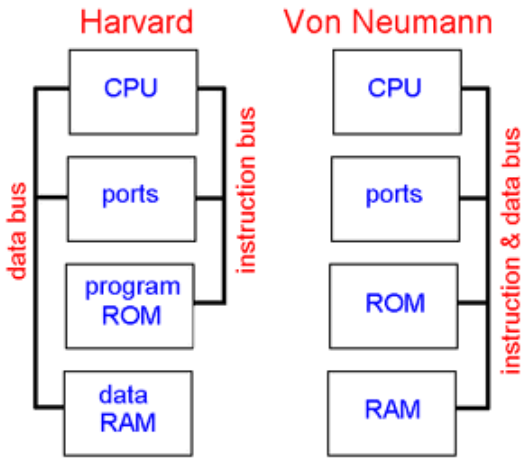
The time it takes to perform an operation is called an **instruction cycle**. An instruction cycle is several clocks long; how many clocks depends on the complexity of the instruction. For fastest possible operation, MCUs are designed to execute instructions in as few clock cycles as possible. The illustration above right shows instruction cycles that take four clock cycles.



Some MCUs use instruction words large enough to also contain data, on the theory that a word that contains both data and what to do with it can be executed in fewer clock cycles. Others **pipeline**, meaning they fetch the next instructions while the current instruction is being executed.

An MCU also has several types of **timers** which count clock cycles and put out a pulse every so often. In many cases, the number of clocks per timer pulse can be set by the user. Suppose you want the MCU to put out a pulse every second. You could write a program that hogs the ALU's computing power, or you could set up a timer to do that, leaving the ALU free to do other things. One type of timer is called the **watchdog**. If regular program execution stops for some reason, the watchdog timer will send a message that says “hey, nothing has happened for a while, something's wrong.” That sets in motion an instruction sequence that restarts the program.

ARCHITECTURE



There are essentially two kinds of computer architecture, originally developed for the earliest electronic computers and still with us today. In **Harvard architecture**, instructions and data are kept separate, like nobility and peasants in old European monarchies. They are usually stored in different memory locations and they travel around the computer on different data buses. **Von Neumann** architecture, on the other hand, is egalitarian and democratic. Binary words are binary words, they're all stored in the same block of memory and moved around on a single data bus, and it's up to the programmer to define what's data and what's program.

The architecture of real MPUs is quite a bit more complicated than the simplified block diagrams I'm showing you. For one thing, there are optional data paths between parts you wouldn't expect to be connected. A person's reaction on first seeing the block diagram is usually “What the hey, this doesn't make any sense.” But providing these options increases the flexibility of the MCU at almost no added cost; all the manufacturer has to do is add a byte-wide gate for each one.

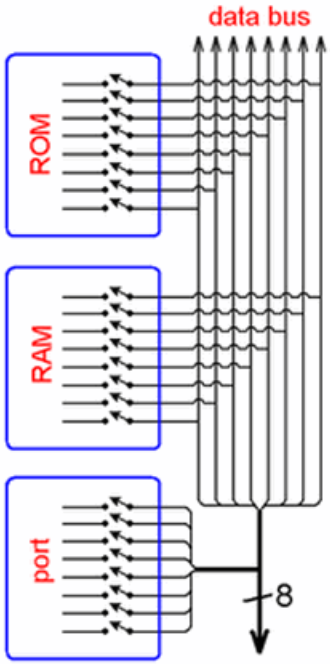
THE INSTRUCTION DECODER

An MCU is a dictatorship, and the program is the dictator. It acts through the puppet dictator, the instruction decoder, that takes a program instruction like “10110110” and translates it into an action. The instruction decoder says what happens and when, which part of the MCU is allowed to use the data bus at a given moment, and which piece of data is retrieved from or stored in which register.

THE ALU

The arithmetic-logic unit or ALU is the brains. All the other parts are involved with storing or routing data. The ALU is where the instructions in the instruction set are actually performed.

THE DATA BUS



If separate wires were provided to send data from any place where it might originate to any place it might be needed, the MCU's structure would look like a ball of steel wool. Instead, there's a single information highway (two in the case of Harvard architecture) to which everything is connected.

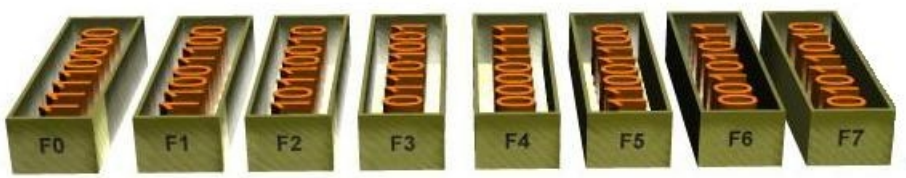
A data bus is like CB radio: people have to take turns. At any given moment, one individual is talking and another individual is listening. The signal goes everywhere, and everybody can hear the message, but it's only intended for one person.

With CB, it's up to each user to play by the rules. In an MCU, the program commands decide who transmits and who receives. Each of the parts connected to the bus has tri-state circuitry as discussed in chapter 2, so two parts don't try to use the bus at once. The fact that the different parts of the MCU have to take turns using the bus is one of the reasons programs are always sequential.

The data bus in an MCU usually has 8 wires, or a multiple of 8 wires, so it can move an entire binary word at a time. In diagrams, the bus may be shown as a single wire, often with a slash and a number that indicates how many wires are actually present.

MEMORY

The memory in an MCU is like a bunch of little boxes, called **registers**, that hold one word each. Each box has an **address**, a binary number that identifies it. When the ALU wants to read the word that's in a particular box, it sends out that box's address, along with a command that tells the box to copy its contents onto the data bus.



When the ALU wants to write data into a particular register, it puts it on the data bus, which sends it everywhere. At the same time, it sends out an address that tells the particular register, “this is for you, store it.” Whatever was in the register before will be erased.

As noted above, an MCU's memory consists of:

- **ROM** or **read-only memory**, where the program and permanent, non-changing data is stored, and
- **RAM** or **random-access memory**, where data that changes is stored. Data in a RAM register, which may change, is called a **variable**.

Let's take a look at each type of memory.

ROM

When an MCU is new, a program is written or **loaded** into ROM. Though ROM can be read any number of times, it can only be written to once; that's why it's called “read-only” memory.

Once a program is loaded into an MCU's ROM, it cannot be changed. Loading a program into ROM is often called **burning** it into the ROM because, in some kinds of ROM, 1s and 0s are recorded by burning out microscopic fuses.

Not everything in ROM is instruction-type data. Sometimes it's data-type data, “variables” that never vary. For example, in an MCU whose only purpose in life is to count down to zero from 82_{10} again and again, the number 01010010_2 will be stored at some ROM address. 01010010_2 isn't an instruction, it's data that the instructions work with, but it never changes so it's put in ROM. That way the user doesn't have to reload it every time they want to use the MCU.

There's no reason instructions and read-only data can't be intermixed, but many programmers keep them in separate “neighborhoods,” so when changes are made to the program during development, there's no need to move the data. Otherwise, when data is moved, the programmer has to go back through the program and change its address everywhere it appears.

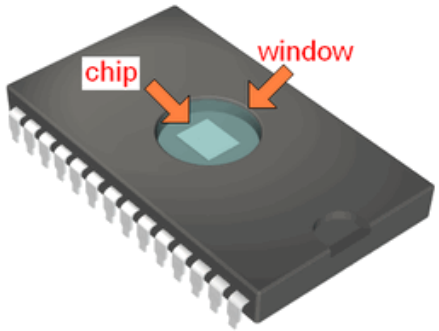
In a manufacturing situation, where hundreds of thousands of MCUs are loaded with the same program, the ROM is programmed when the MCU is made. This is done photographically; the ROM is exposed through a pattern called a **mask** which is a lot like the negatives used to make regular photographs, and similar to the masks that are used to make the transistors and traces in the MCU. This is called **mask programming**.

It takes a lot of money to set up a mask program, which makes mask programming too expensive when fewer MCUs are going to be produced. So, in smaller quantities, MCUs are individually programmed, electrically. This is done by placing a new MCU into a programming device called a **burner** that transfers a program from a computer into the MCU's ROM one instruction at a time. The ROM is told to "learn" an instruction by putting a voltage on a programming pin, or in some cases by sending a "learn" instruction to a port. Once the program is loaded, many MCUs have a function that allows the user to "burn the bridge" to the ROM, preventing accidental (or malicious) alteration of the program, and also preventing the program from being read out, so others can't copy it.

Rom that is programmable by the user is called **PROM** or **programmable ROM**.

During program development, it's useful to be able to erase and re-load a program. After all, the programmer seldom gets everything perfectly right on the first try. MCUs are the ultimate rule-following geeks. If there's even a tiny flaw in the program, the MCU doesn't know what to do and everything grinds to a halt. (That's called a **crash**.) So the program is likely to be adjusted and loaded into ROM a number of times before it runs smoothly.

Erasable PROM is called **EPROM**. Some IC packages have a window over the chip. Strong ultraviolet light will erase any program that's in ROM, allowing it to be reprogrammed. In normal use, an opaque label is glued over the window to prevent accidental erasure.



In other MCUs, the program can be erased electrically; this is called **EEPROM** for **electrically erasable PROM**. MCUs with erasable ROM have traditionally been special-purpose devices used mostly for development, much more expensive than MCUs with regular write-once ROM. So much more expensive, in fact, that many programmers just grab a handful of regular MCUs and keep throwing them away until they get the program right.

Program glitches are called **bugs**, and the process of perfecting a program is called **debugging**. You may have heard the story that the use of the word "bug" to mean "defect" originated in the days when computers used relays instead of tubes or transistors. A malfunction was traced to a bug, specifically a moth, trapped in one of the relays. The

moth was taped to a note explaining the problem, and note and bug are now in the Smithsonian. It's such a great story that I wish it were entirely true. But, though the moth and the note are real, and they are in the Smithsonian, Edison talked about electrical bugs in the 1870s. JFYI.

RAM

Data which isn't permanent is stored in RAM, which is often called **read/write** memory because you can write data in and read it out as many times as you want. A RAM register is one binary word of clocked latches. Reading and writing is controlled by instructions from the CPU.

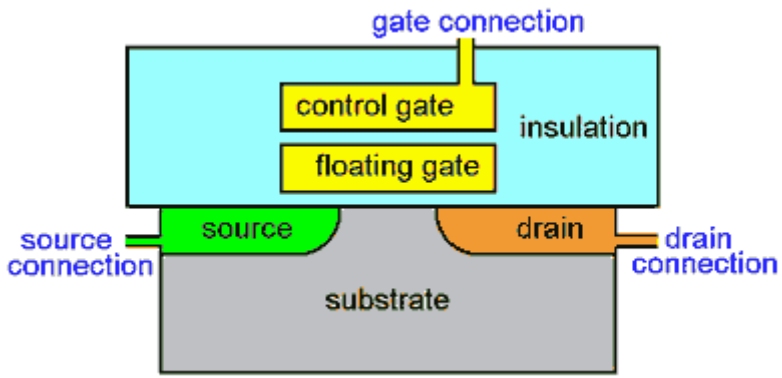
MCUs usually have a lot more ROM than RAM, because there's usually a lot more program and fixed data than variable data. Think of the MCU that would run a washing machine. The program would consist of many steps, with many different events that must be timed and decisions that must be made depending on wash cycle selected and so

on. Variable data, on the other hand, is limited to a few bits that represent the water level, a bit that tells the MCU when the load is unbalanced, and a byte or two for the buttons that the user pushes to select the cycle.

RAM Volatility

RAM is **volatile**, meaning data is only retained as long as the system is powered up and is lost when it shuts down. If necessary, RAM can be made **nonvolatile** by providing a battery or large capacitor to keep the memory alive even when power is shut down. CMOS memory draws almost no current when not actively switching, which of course it won't be when the main power supply is off.

Flash memory is a newer technology with characteristics of both RAM and ROM. It's based on a special kind of MOS transistor that uses the charge trapped on a floating, insulated gate to close or open the channel. Flash is almost as rewritable as RAM – you can't rewrite it an infinite number of times, but you can rewrite it thousands of times.



flash memory FET

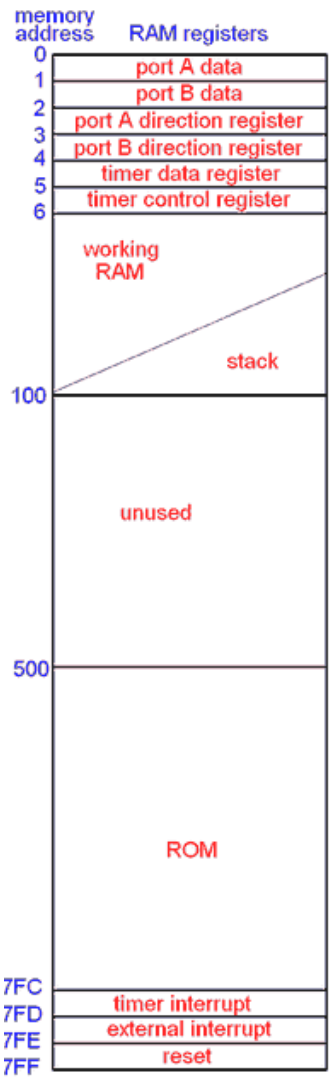
And it's almost as nonvolatile ROM – the data will eventually fade away, but it will take decades. Since that's longer than the useful life of most equipment, flash is essentially permanent. Flash technology has given us a new kind of ROM, the **EAPROM** or **electrically alterable PROM**.

In normal PROM, all the data must be erased before new data can be loaded. With EAROM, individual registers can be changed. EAROM is essentially nonvolatile RAM, except that there's a limit to how many times you can change the data before the chip starts to malfunction.

THE MEMORY MAP

With the exception of the accumulator and some of the support registers, every RAM and ROM location has an address. In most MCUs, lower addresses are assigned to RAM and higher addresses to ROM.

Let's work with an imaginary MCU, the ETA1234. Suppose our MCU has 256 bytes of working RAM and 768 bytes of ROM. The addresses may not go directly from RAM to ROM, because many MCUs come in families that range from simple to complex. It's nice to be able to write a program that will run on any of them. So RAM may always start at address $00 (hex 00, or 00_{16}) and go as high as it goes, in this case to $FF or 255_{10}. ROM might start at $0500 ($1280_{10}$) and go to $07FF (2047). That gives us 256 bytes of RAM starting at $00, a run of addresses between $0100 to $04FF with no memory, and 768 bytes of ROM beginning at $0500. Some addresses are preassigned to specific functions, such as ports.

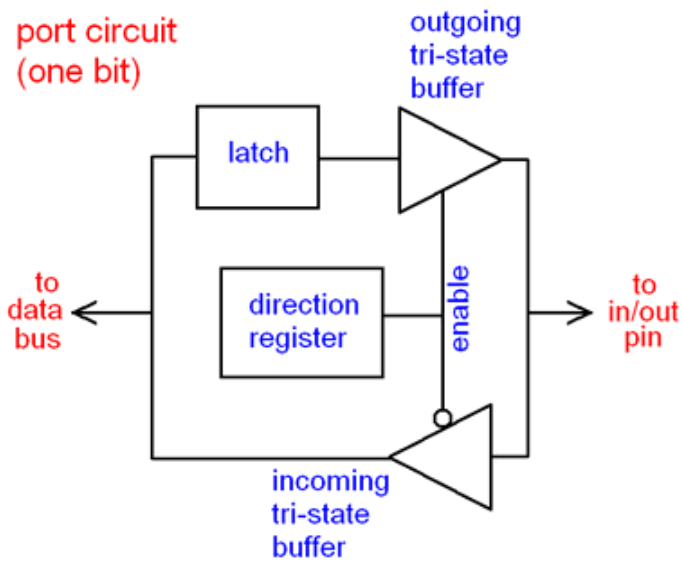


In some MCUs the **stack** is part of working RAM. Recall that the stack is what keeps track of where the program counter was when it jumped to some other place. A multiple-level stack allows jumps from jumps, which is more common than you'd think. Shared stack addresses start at the end of RAM and go backwards; this kind of sharing lets the programmer use each register for working memory or stack as required. At the other end of town, the highest ROM addresses usually contain a few pre-loaded instructions that tell the MCU what to do if, for example, a power supply glitch throws the program counter off track and things come to a halt. The highest address, often called “reset,” is where the program counter goes when power is first turned on.

At this address, the user stores another address which tells the MCU where to go to find the beginning of the real program.

PORTS

If an MCU were an island, the data bus would be a highway that goes right to the edge of the water, permitting ships to carry cargo (data) to and from other places. After all, an MCU isn't of any use unless it can communicate with the world at large. This is done via **ports**, which are connecting pins that allow data to be passed in and out. Most MCUs have **parallel ports**, with a wire for every bit, so an entire binary word can go in or out at the same time. Many also have **serial ports**, where a binary word is clocked in or out one bit at a time. The smallest MCUs have only serial ports because there aren't enough pins for a parallel one. In most cases an MCU port is **bidirectional**, meaning that it can be an input one moment and an output the next.



When a port is set to output, it's essentially a regular RAM register except that the bits are connected to pins, so the outside world can read what's in them. In other words, when data is read into it, it latches, and that data will appear at the port until new data is written over it, or the port is changed to an input.

In most MCUs, the ports are **memory mapped**, meaning that a port has an address like any other register in RAM. Think of a row of houses, each with its own address. The woman who lives in the first house on the left runs a jewelry import-export business from her home. If you want to send jewelry overseas, you mail it to her address and she exports it. If you want jewelry from overseas, she imports it and mails it to you. In computer terminology, she's a memory-mapped port; her address, and her mailbox, are the same as everybody else's. She just happens to live at the water's edge and have a dock in her back yard.

Ports tend to be located at the lowest RAM memory addresses. Right next to them, we usually find **port direction registers**, which control whether they're inputs or outputs at a particular moment. (Look again at the ETA1234 memory map on the previous page.) Each parallel port needs a whole byte of direction registers because the port's pins can be individually set to input or output. In the illustration at left, note that the incoming gate is an example of why an active-low state may be useful: any time the outgoing buffer is disabled, the incoming buffer is enabled.

PROGRAMMING

Okay, we've got all the hardware. Now let's look at how a program is written. I'm not going to teach you how to program. That's too big a subject for this study guide, and you'll want to choose an actual MCU before you do start to program.

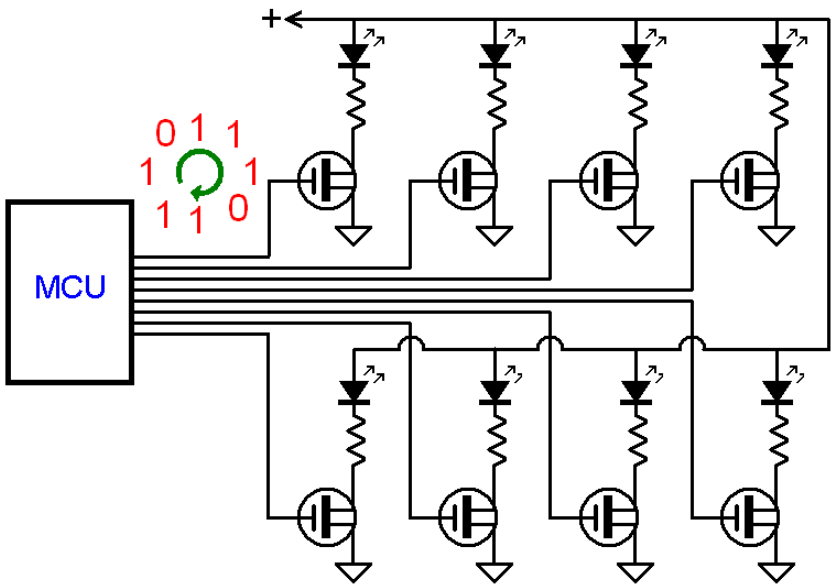
Real programs follow strict rules, and a particular instruction might be represented by 11011100 in one MCU family and 10100011 in another. If I were to write the following in the actual programming language for one kind of MCU, you'd find it a stumbling block if your own first hands-on experience is with a different MCU. So I'm just going to write generic instructions.

Programs consist of lines of instructions called **code**, usually one line per instruction. If I were writing a real program I would be a **programmer** or **coder**.

Let's go with an example I used in a previous chapter, the Las Vegas chase-light sign. This could be done with just a shift register, but let's do it with an MCU instead. We have to set up the hardware first. Let's assume the MCU has an 8-pin output port. An output of almost any kind will have a pulldown transistor, and most will have at least a pullup resistor, if not a transistor. In some MCUs, pullups can be enabled or disabled by program instructions. I'm going to assume each port has a pullup resistor.

Each pin of the port is connected to a power MOSFET in common-source mode. LEDs are connected between the drains and a positive voltage source, which is probably different from the source that's powering the MCU. After all, high-quality voltage filtering and regulation costs money, and LEDs don't need it.

Since a MOSFET's gate is an open circuit, even if the port's pullups are weak they'll be okay. There is one potential issue, though. Say the MCU runs on +5V, which means a 1 at any output will be 5V. Not every MOSFET is turned fully on by a 5V gate voltage.

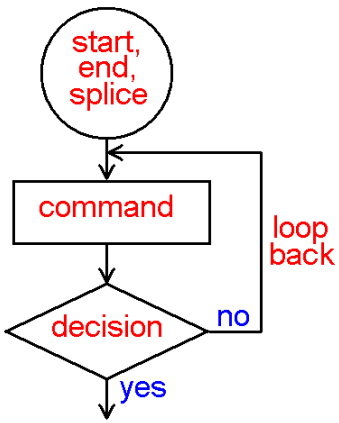


Since a MOSFET's gate is an open circuit, even if the port's pullups are weak they'll be okay. There is one potential issue, though. Say the MCU runs on +5V, which means a 1 at any output will be 5V. Not every MOSFET is turned fully on by a 5V gate voltage.

If we want the MOSFET to act like a switch, we need to look through the Mouser catalog and choose a MOSFET that is compatible with 5V logic. You can download complete specs on any likely-looking candidate.

You don't have a Mouser catalog? Get one, they're free. One of the important steps in mastering this stuff is paging through catalogs to learn what's available. Also look at DigiKey (online only), Radio Shack online, and Element 14. (So named for Silicon, element 14 in the periodic table.)

THE FLOW CHART



Before a programmer ever starts working with 1s and 0s, they write out the logical flow of operations in a **flow chart**. This is a diagram with instructions written inside geometric shapes and lines that show how the program flows from one instruction to the next. Software is available to help the programmer develop flowcharts on a desktop computer, but many programmers find a pencil and paper easier.

The most basic instruction is **start**, signifying the beginning of the program. Start and end points are listed in circle or oval shapes.

Unconditional commands, those that are simply executed in sequence, are shown in rectangular boxes. Ideally, these boxes line up in a column, with instructions flowing from top to bottom. Real programs are usually too complex for things to work out that neatly.

Conditional operations, where a decision must be made and the next step depends on the outcome, are listed in diamond-shaped boxes. Program flow comes “in” at one of the points, usually the top; the other points represent possible outcomes, which are labeled.

The programmer has complete flexibility when designing a flowchart. Any instruction can lead to any other instruction. It can even loop back to an earlier point in the program.

We may not need an actual “start” instruction. When an MCU is first powered up, the program counter goes to a predetermined memory address to find the first instruction. As mentioned above, in many MCUs, that address is the very last address in memory, $07FF in our ETA1234 example. The counter can't count any higher than that, so that address must store a command to jump to a lower memory address where the real program is located.

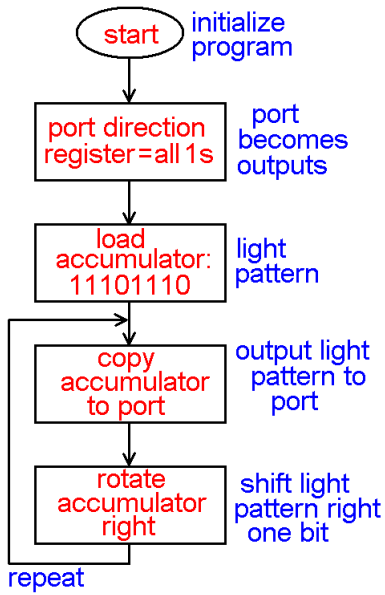
In our chase-light program, the first command will be to turn the port into an output, so data can be sent to the MOSFETs.

The second command will be to load a binary word representing the light pattern into the accumulator.

Suppose we want a pattern of three-lights-on, one-light-off. The word that represents that is 11101110. That's six 5V outputs and two 0V outputs. If you connect those to the gates of eight MOSFETs, you'll get a LED pattern of three on, one off, three on, one off.

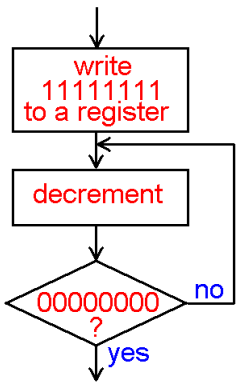
The third command will be to read the word in the accumulator into the port, which will latch as if it were RAM. After this command is executed, the accumulator will still have the original word in it.

The fourth command is to shift the bits in the accumulator to the right. If we just shift, we'll lose the right-hand bit and after eight shifts all the lights will be out. So we have to use a command called “rotate” where the right bit comes around and re-enters the accumulator at the left end.



From this point, the MCU only needs to keep on doing what it just did: shifting and outputting. So the fifth command will be to jump back to the third command. This will set up a loop that continues indefinitely. Note that there's no “stop” instruction; it doesn't end until the power is shut off.

Are we done? Nope. In most cases, the clock frequency is so high that the pattern will scan too fast for us to see the movement. We have to insert delays between shifts to deliberately slow the program down. So, let's insert a time delay between “copy accumulator to port” and “rotate accumulator right.”



A time delay can be implemented by putting a number in a register, then decrementing it until it reaches zero. The command sequence is: **Have we waited long enough yet? / Nope. / Okay, wait a little longer. (repeat) Have we waited long enough yet? / Yep. / Okay, on to the next part of the program.** In other words, this is a time-wasting loop.

Note that we now have a loop within a loop. The large loop is the shift-and-output loop. The small loop is the decrement loop, which slows down the large loop.

You're thinking, “couldn't we just use a slower clock?” Maybe. But in many MCUs, the RAM is so volatile that a slower clock would allow the data to be lost between **refreshes**, which we'll talk about in the next chapter. Most MCUs will work with a clock as low as 32kHz; lower than that, yer on yer own.

Some MCUs are so dependent on the accumulator that the byte representing the LED pattern would have to be briefly stored in memory during each loop to free up the accumulator for the decrement loop. In this particular case that wouldn't present a problem, since we need the delay and we've actually had to go out of our way to slow things down. But where high speed is important, that would be a stumbling block. As you learn more about specific MCUs' architectures, you'll see various enhancements to avoid this kind of problem, one of which is multiple accumulators.

By the way, did you notice? The ROTATE instruction turns the MCU into a ring counter, which we studied in chapter 4.

THE LOOKUP TABLE

There's another way to generate a repeating pattern: instead of calculating it step by step, simply program all the binary values into a block of ROM and then read them out sequentially. For example, we could take four addresses and put the following into them:

address A	**11101110**
address B	**01110111**
address C	**10111011**
address D	**11011101**

Now all we have to do is loop through these four addresses, again and again, transferring their data to the port, and we'll get the Vegas chase lights. No computation involved.

This is an example of a **lookup table**. It's especially useful when computation would take too long. For example, it's quite a chore for a computer to compute a sine wave on the fly, especially at high frequencies. But it's easy to step through a lookup table. If you did the EM3 study guide, you remember that a tuned RC sinewave generator has low distortion but is hard to stabilize and a function generator is stable but has high distortion. A lookup table can be used to build a digital sinewave generator that is both stable and clean.

In math operations it is often easier to provide a lookup table than to calculate, as in finding a logarithm. The user enters a number which is used to access the memory address that holds the corresponding logarithm. This would be a big lookup table, but ROM is cheap.

MACHINE LANGUAGE

Back to our chase lights: once the programmer has a flowchart, it's time to translate the series of steps into instructions in an MCU's ROM. But ROM, being binary, can't store English words like “rotate right.” It can only store 1s and 0s. So each instruction has to be translated into a binary word that the CPU will understand as a command. Instructions in binary form are called **machine language**, the language the MCU understands. Let's say 00101100 is an instruction to rotate the data in the accumulator to the right. We think of this as the number $2C, but it isn't really a number at all, it's a positive voltage on three of eight program wires within the MCU. A rather complicated gate logic circuit in the instruction decoder interprets that pattern and directs the ALU to perform a shifting operation.

(I'm not saying 00101100 is actually a rotate right instruction; I just pulled that number out of the air. Each family of MCUs has its own set of binary instruction codes, and they're all different.)

Early computers, including the first personal computers in the 1970s, required the programmer to write code directly in machine language. Instructions were punched into paper cards or entered one at a time via a bank of switches. A programmer would keep a list of instructions and their related machine codes, e.g. “rotate right = 00101100.” The illustration on the next page is an early personal computer from Altair – note the group of eight switches on the front panel. (Image from Wikimedia Commons)

Some people still insist on working in machine language. They like to know exactly what's happening in every single register. Nowadays, however, they write the whole program into a file in a desktop or laptop computer. When the program is complete, they plug the MCU into a **burner,** which burns the program into the MCU's ROM. A much more practical way to do things.

Writing the program into a computer file has another advantage. You can put an app on a laptop or desktop computer that's a virtual MCU. This is called an **emulator** and it's a program that mimics a particular MCU's functions. The program can be run in the emulator (usually at much lower speed) and bugs can be found and fixed. Then, once the program is fully debugged, it can be transferred to the MCU.

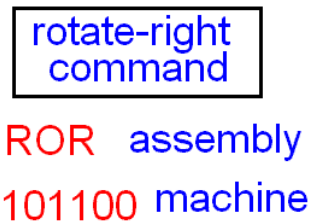
You can learn a lot about programming before you ever get your hands on a burner and MCU. An emulation program works exactly like an MCU, just slower, and you can usually find an emulator for a particular MCU for free. You can even get emulators for different MCU families to learn each family's strengths and weaknesses.

ASSEMBLY LANGUAGE

Once we're using laptop and desktop computers to to develop MCU code, we can go one better and make it easier on ourselves by using an **assembler** to help us write the code.

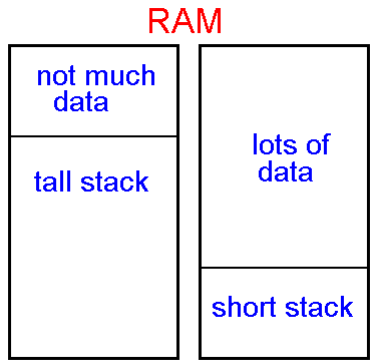
An assembler is a translation program. In **assembly language**, each instruction in the instruction set is identified by an English word or a few letters that are much easier than a binary word to understand and remember.

For example, “ROR” may mean “rotate right.” (I say “may” because each family of MCUs has its own assembler and the command designations are not always the same.) When you say “ROR” to an assembler, it says “00101100” (or whatever the machine-language command is) to the file where the program is being assembled. This means a single line of assembly code, containing a single **mnemonic** like ROR, corresponds directly to a single binary command that will eventually wind up in the MCU's ROM. If you thought “lookup table” you get five points. Thanks to assemblers, a programmer never needs to know or care what the actual instruction 1s and 0s are. Once the program is complete in the computer and debugged, it's transferred to the MCU. Most assemblers provide additional services, like keeping track of memory addresses so the programmer doesn't have to. A programmer could invent a name for a piece of data, like “interest rate.” The assembler would decide which address to store that data in, and then would know when the programmer called for “interest rate” where to find it.



One useful assembler function is dealing with complex MCU architecture. Some MCU families began life as simple little chips, and have morphed into complicated big chips. Like a house that's been added to many times, their architecture is a patchwork of adaptations that would confuse a machine-language programmer. But a good assembler will keep track of where things are and present a sensible, logical user interface to the programmer.

One of the most important things a good assembler does is prevent blocks of data from trespassing on each others' territory. For example, we've noted that the stack pointer may share RAM with other data. This allows the programmer to write a program with a short stack and a lot of data or a tall stack



with a little data; in other words, it makes best use of limited RAM. But if we start adding data at one end of the RAM and a bunch of jumps in the stack pointer at the other end, we could have a collision in the middle where they're overwriting each other. A good assembler will warn you if you're about to do something that will cause a problem.

HIGH LEVEL LANGUAGES

The next step above the assembler is the **high level language**, e.g. **C++**, that takes things a couple of steps further. There are two fundamental differences between a high level language and assembly language.

First, a single line of high-level code may translate to a whole series of binary instructions that accomplish a certain thing. Suppose you're in a restaurant and you want to order a pepperoni pizza. There are a number of steps involved:

1. **knead the dough**
2. **roll it out**
3. **put on the sauce**
4. **put on the cheese**
5. **put on the pepperoni**
6. **put the pizza in the oven**
7. **let it bake for a while**
8. **pull it out**
9. **cut it into pieces**
10. **put it on the table in front of the customer**

]Suppose we were to write those steps in assembly:

1. **KND**
2. **ROL**
3. **ADS**
4. **ADC**
5. **ADP**
6. **OVN**
7. **BAK**
8. **RMV**
9. **CUT**
10. **SRV**

But when you walk into a restaurant and order a pizza, you don't hand the waiter a list with those instructions. You order the pizza and assume the employees know what to do.

That's how a high-level language works. You write a single instruction...

Make me a pepperoni pizza

...that the high-level language translates to perhaps hundreds of machine-language steps.

The second difference between assembly and a high-level language is that, while a given assembler works only with a particular family of MCUs, a high-level program is MCU-independent. It works with any MCU that has the necessary instruction set to get the job done – just as an order for pepperoni pizza will get you a pepperoni pizza in any restaurant equipped to make one.

Many high-level language names are acronyms that suggest what they do. One of the earliest was **Cobol**, or **co**mmon **b**usiness-**o**riented **l**anguage. The “bol” ending inspired imitations like Snobol and Futbol. Some were real, others were jokes. I was disappointed to learn that Dirtbol is a real language. It should have been a joke language where the computer made raunchy comments about the user's intelligence, romantic potential and personal hygiene. Maybe somebody will do it yet; I think Scumbol is available.

One of the more popular languages is **Basic**, which is no accident. Basic was specifically written for ease of use at a time when most people were still wrestling with machine language. The name “Basic” was chosen first and then words were rather clumsily fitted to it: **Beginner's All-purpose Symbolic Instruction Code**. Despite the reference to beginners, Basic and newer variants like **Visual Basic** are widely used by programming professionals.

One very influential and widely used language is “**C**,” which goes all the way back to Bell Labs and 1969 and is based on an even older language called “B.” Variants include **C++**, and others like **Java**, **JavaScript**, **Perl** and **Python** have been borrowed from it. Someone who wants to seriously get into programming could do a lot worse than learn C.

COMPILERS

When it comes time to load up the MCU, the high level language must be translated to machine code. The translator is called a **compiler**, and it's a program that runs on a desktop or laptop. The compiler is the cook who takes the pizza order and translates it into the individual steps (machine code) needed to make it.

Different restaurants have different ovens, so the time it takes to bake a pizza will vary from one kitchen to another. Likewise, different MCU families have different instruction sets, so different compilers are needed to translate a program written in a high level code to machine code.

INTERPRETERS

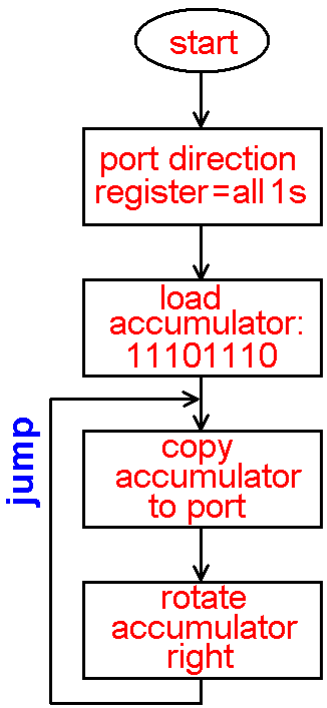
One problem with downloading an application from the internet is that it may be written in a language your computer doesn't support. The original purpose of **Java** was to permit the necessary execution code to be downloaded right along with the application. In Java, translation from high-level code to machine code is meant to be done in real time. A compiler that operates in real time is called an **interpreter**.

High-level languages do not result in efficient machine code. The most efficient, most compact, fastest-running code results when a skilled programmer writes machine code. A program written in assembly will be a little larger and a little slower, and a program written in a high-level language largest and slowest.

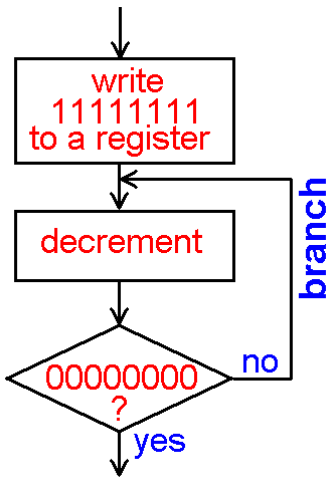
In most applications, this doesn't matter, because a program usually doesn't bump up against an MCU's limits. When it does, it may be more cost-effective to use a bigger MCU than optimize the program. Programmers' time is expensive.

JUMP AND BRANCH INSTRUCTIONS

As I said way back up there so far I don't remember where, the natural behavior of the program counter is to chug through the register addresses in sequence: F05, F06, F07 F08 and so on. But now and then it will encounter an instruction that tells it to jump to some other address.

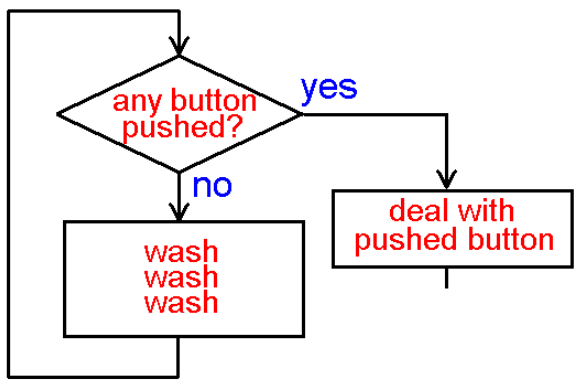


In the "program" I wrote for chase lights, there are two cases where the program jumps back to a previous instruction. At the end of the main loop, the counter is told to go back to a previous step. This is a simple **jump** instruction.



The decrement loop includes a step where a decision is made, represented by the diamond shape. After each decrement, the loop counter is tested to see if it has reached zero. If it has – if the test result is a "yes" – the program counter proceeds on to the instruction at the next address. However, if the loop counter has not yet reached zero – if the test result is a "no" – the program counter is instructed to loop back to the decrement instruction. This is a program **branch**, like a tree branch that sticks out from the trunk, a side road that's a detour off the main road. A branch instruction tells the program counter, "a special event has occurred, go to this address instead of where you were headed next." In this example, the branch represents the normal case; it occurs 255 out of 256 times. There's no law saying that's taboo.

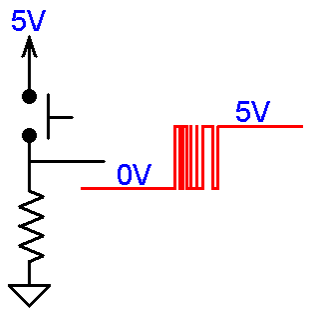
Another example would be a pushbutton that diverts normal program flow. Consider the MCU in a washing machine. Each of the pushbuttons – normal cycle, delicate cycle, hot, warm, cold – might be connected to a pin of one of the MCU ports. 99.9% of the time, none of them are pushed. As the MCU goes about its business, it periodically checks the port. The flow chart would have a diamond at that point in the program that asks the question, "is any bit in the port set?" 99.9% of the time, the answer will be no, and the program will proceed normally. But once in a while, the answer will be yes, indicating that a button is being pressed. That will trigger a branch instruction, telling the program counter to jump to a memory address where instructions are stored to deal with a pressed button.



SUBROUTINES

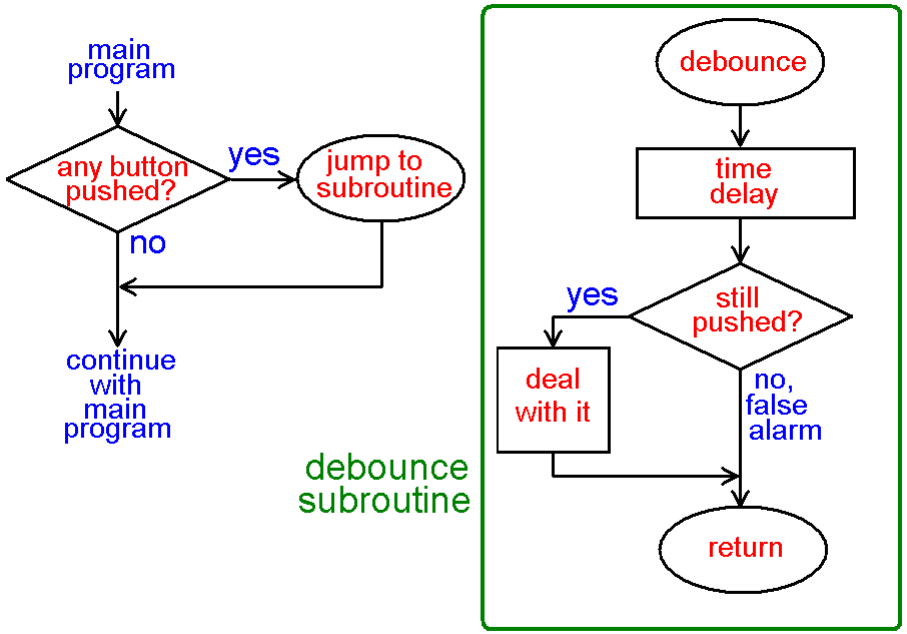
A particular instruction sequence may show up many times in a program. In such cases, it makes more sense to create a single, separate mini-program with those instructions. Then, whenever that sequence needs to be executed, the main program jumps to the mini-program. Mini-programs of this type are called **subroutines**.

One common use for a subroutine is to **debounce** switch contacts. When mechanical switch or pushbutton contacts close, they bounce, closing and opening a number of times in the first few milliseconds before closing for good. If the pushbutton is operating a relay or doorbell, this doesn't matter. But an MCU is so fast that it interprets each bounce as a separate closure.



Suppose the pushbutton is a channel changer on a TV remote. If it goes up or down five to ten channels at a time, the user really can't control it. So, to keep this from happening, the programmer writes some debouncing code, which is an instruction sequence that doesn't output a 1 until it's sure a button is really closed, and then doesn't output an 0 until it's sure the button is open again. A TV remote may have twenty buttons, and every button must be debounced. It doesn't make sense to put twenty copies of the debounce instructions at twenty places in the main program. Instead, it's written once as a subroutine, and the main program calls it up whenever any button is pressed.

In a flow chart, the start and end points of a subroutine are identified by round or oval shapes, just as in the main program. Any time you have a connection point where part of the program attaches to another point, round/oval shapes are used as in the illustration at right.

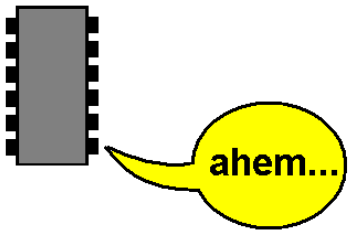


What happens when the subroutine is finished? There are two options. In one case, the last command in the subroutine tells the program counter to go to a specific, predetermined place in the main program. In the other case, the last command tells the program counter to go back to wherever it came from – that is, back to the address of the next command in the main program. But if the subroutine is used many places in the main program, how does it know where to return to? It doesn't. That's the stack's job.

THE STACK

Now that you've been introduced to subroutines, you can better grasp the purpose of the stack. Before the main program jumps to a subroutine, it makes a note of where it is by putting the address of the current instruction in the stack. Then, when the subroutine issues a “return” command, that address is pulled out of the stack and the main program resumes from that point. Essentially, the subroutine says, “Now, where were we?” and the stack answers, “Right here.”

INTERRUPTS



It's a pretty big job for the MCU to keep checking on everything that's going on. So it says to everybody around it, “Look, I'm not going to constantly check up on you, I'm just going to follow my normal routine. If anything unusual happens, bring it to my attention.”

The “bring it to my attention” function is called an **interrupt request**, and the MCU usually has one pin dedicated to it. Whenever that pin goes high (or low, depending on the MCU), the MCU stops what it's doing and **services** the interrupt. Usually, this means jumping from the current place in the program to a subroutine. If the program is doing something critical, the programmer can set a **mask interrupt** bit that blocks the interrupt until the program gets to a good stopping point.

An interrupt request initiated by the interrupt pin is called a **hardware interrupt**. There are also **software interrupts**, generated within the MCU, that can be activated by program commands. For example, one of the timers may be set to interrupt the program every so often.

A **non-maskable** interrupt is an interrupt that the program must service immediately, whether the mask bit is set or not. For example, the MCU in a machine tool would receive a non-maskable interrupt when a sensor showed that the operator's fingers were where they shouldn't be.

Since I've brought up machine tools and safety, this is an appropriate place to mention the relationship between programming and fail-safe operation. There's a very good chance your career will take you to a place where you're dealing with MCU-controlled machinery, and perhaps writing the code. It is not enough to write code that executes properly when everything is as it should be. You will also need to include code to detect every malfunction you can imagine and stop the machine immediately.

B-movie robots that lose their minds and start killing people are not fiction. Any big machine can do that if the program in its MCU or the sensors connected to it are messed up. The reason you seldom hear of such things happening is that machine designers provide software and hardware safeguards, and backup safeguards, and then another layer of safeguards on top of that. Remember Murphy's law: if there's any possibility of something going wrong, you can bet it will.

One very useful safety tool is the watchdog timer, which is always just a fraction of a second from timing out. If anything goes wrong, and normal program execution is disrupted, the watchdog timer will stop everything within a very short time – especially if the program has been written with this in mind.

OTHER FUNCTIONS

What I've described up to this point is the very most basic MCU as it existed in the mid-1970s. Many of those simple chips are still on the market, but newer designs incorporate many features that reduce program complexity or the number of things that must be outsourced to external circuitry.

MULTIPLE ACCUMULATORS

Modern MCUs have up to 32 accumulators. This tremendously speeds up programs where more than one variable must be dealt with. Coupled with optimized, pipelined architecture that executes an instruction every clock cycle, this permits some MCUs to do complex tasks at a rate of 100 million instructions per second.

Some MCUs have no accumulators at all; instead, the whole RAM is available for accumulator use. That maximizes flexibility but may reduce speed.

ANALOG TO DIGITAL and DIGITAL TO ANALOG CONVERTERS

We're going to spend all of chapter 7 talking about how analog signals can be represented in digital form and the things that makes possible. A simple task for an MCU with an analog to digital converter would be to convert the output of an analog temperature sensor to digital data and use that to control a furnace or air conditioner. Every programmable thermostat with a digital readout has an MCU with this capability.

ANALOG COMPARATOR

A comparator has a 1 or 0 output depending on which of two input voltages is higher. An MCU that incorporates a comparator also usually incorporates a precision voltage reference, which makes the comparator easy to use.

PULSE WIDTH MODULATED OUTPUT

This is a rectangular-wave output whose duty cycle is controlled digitally. One application is to convert digital data to sound. A PWM output will directly drive a piezoelectric earpiece of the kind used in cellphones.

UART

This requires a bit of discussion. Moving data between MCUs is often a complicated business, especially if a parallel-to-serial or serial-to-parallel transformation is required. The gizmo that handles this is called a **UART** or **universal asynchronous receiver-transmitter**.

The meaning of “receiver-transmitter” is obvious enough, that's just a bidirectional port. “Universal” just means that

the UART is designed to communicate with a wide variety of other devices. But what's that “asynchronous” business?

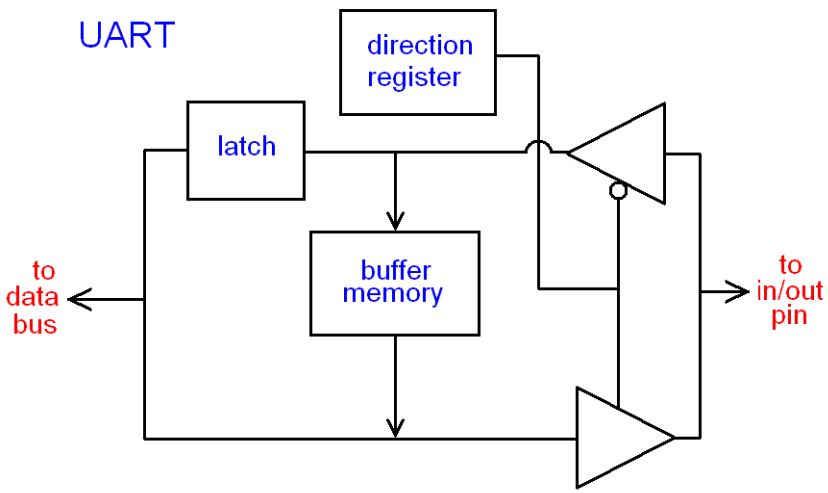
As American tourists abroad are often rudely surprised to discover, American laws mean nothing beyond our borders. Other countries are sovereign nations and when you're on their soil, their laws apply.

In the same way, within an MCU the CPU rules everything with an iron fist, but it doesn't rule the external devices connected to its ports. Those devices have their own CPUs and programs, and their clocks aren't synchronized with our clock. So our MCU can't just send a byte to a port and know that it will get where it has to go, because the other device may not be ready to receive it or may not speak our MCU's language. There's some negotiation that has to go back and forth, saying, “data is ready to send” and “I'm ready to receive it” and “okay, I got that word, send the next one” or “I didn't get that, send it again.”

These are called **handshaking** functions, and they're necessary to insure that data doesn't get mangled during the transfer. What this means is that the predictable timing of normal MCU operation is lost when it's communicating with other MCUs. Each thing happens when it happens, and we can't control the timing; the best we can do is make sure things happen in the right sequence. What's why a UART is “asynchronous.”

All this handshaking means data going in or out often has to wait. To avoid bogging down the program, the UART has some RAM of its own, called the **buffer.** This is a different meaning from a Boolean buffer. A data buffer is a bit of memory, a holding room where data waits to go through customs.

The CPU sends outgoing data to the buffer and then goes on doing what it needs to do; meanwhile the UART takes data out of the buffer and exports it at whatever rate the handshaking permits. Incoming data can also be stored in the buffer if the CPU is in the middle of an interrupt-masked process.



The UART can be a separate chip, or a virtual UART made of instruction code, but many MCUs now include a hardware UART that makes this complicated business easy and cheap.

The UART diagram to the left shows the re-timing function, but doesn't show the translation function. One of the most important things a UART may do is take data from the data bus and change it to a different format for transmission, or receive data in a different format and convert it back to the local language. These standards are often called communication **protocols**. The UART is like a diplomat who knows many languages and cultures and helps the CPUs in different units communicate.

REPRESENTATIVE MCUs

At this point, we've skimmed enough of the surface that you have a general understanding of what an MCU is and what goes on inside. When you have time to kill, or get so tired of my blather you'd sooner whack your kneecaps with a wrench than read more of it, go online and study the architecture and instruction sets of various MCUs. Three I'd recommend:

- The 6800 series, originally made by Motorola when they were a semiconductor manufacturer, spun off in 2004 into a new company, Freescale. The main virtue of the 6800 series, at least to me, is its simple, easy to understand architecture. Simple is good. We like simple. Let's hear it for simple. Yay, simple.

- The Z8 and Z80 from Zilog. One of the companies credited with inventing the microcontroller is Intel, who first made the 4004, then the 8008, then the 8080. At that point development split: Intel developed the 8080-derived 8086, then 286, 386, 486 and Pentium, while one of the original 8080 developers went off and founded Zilog and the Z-MCUs. These dominated the small-computer market in the late 1970s and early 1980s. Their main virtue is an almost 40-year track record and the near-certainty that they aren't about to disappear. I wish I could say the same for other MCUs I've designed into equipment, then found I could no longer get.
- The PIC series from Microchip Technologies. General Instruments is one of the other companies credited with inventing the microcontroller; Microchip was a spinoff. The PIC's main virtues are the great variety of PIC chips available, the broad user base, and the fact that Microchip has always been friendly to students and small-quantity users. The architecture is clumsy, so don't try to program in machine language. Get an assembler.

Don't just limit your study to these three examples. By virtue of the fact that they are old, well-established MCUs, they take a back seat to other, newer designs in many ways (though their manufacturers do keep them updated). Mouser, Digikey and other suppliers carry a wide range of MCUs, and you can download detailed information directly from their sites, or from the manufacturers. Just looking at the different architectures and instruction sets will give you ideas about ways to do cool things.

MODULES

There's a middle ground between the MCU and the full-fledged computer, and that's the module with an MCU and enough support circuitry to do some semi-advanced things like machine control. The **Arduino** system has been getting a lot of press, especially because there are a whole slew of related modules you can attach that extend its capabilities. But it's by no means the only option. The same suppliers that sell MCUs sell modules too, so check them out while you're surfing.

End of Chapter 5

EM4 Chapter 6
COMPUTERS

Objectives of this chapter:

- *discuss the differences between a computer's MPU and a typical MCU*
- *define “peripheral” and give examples*
- *define “mother board” and explain why it's called that*
- *give examples of ROM and RAM solid state memory inside and outside the MPU*
- *discuss the operation of a magnetic hard drive*
- *discuss the operation of a DVD or CD ROM drive*
- *explain basic error correction techniques*
- *define “daughter board” and give common examples*
- *discuss simplex, duplex, half duplex communication*
- *describe 1-to-1, star, multidrop, daisy chain designs*
- *describe how an RS232 system operates*
- *describe how an RS485 system operates*
- *describe USB and name the connections*
- *describe the operation of a MIDI system*
- *define and discuss “LAN” and “WAN”*
- *describe an ethernet system*
- *describe Firewire and Thunderbolt systems*
- *explain matrix scanning of a keyboard*

You've already learned a lot about how a computer works, because at the heart of a computer is something that's a lot like an MCU on steroids. A computer does the same things in the same way, but a lot more of them, a lot more flexibly and a lot faster. Compare a single book to Wikipedia. That's how an MCU compares to a computer.

In addition to the computer itself, a computer system also incorporates things that are attached to it, like keyboard, printer, monitor, speakers, mouse etc. When they're not built into the computer, they're called **peripherals**. At first I had thought of putting peripherals in a separate chapter, but I realized that there is no reliable dividing line. Most of the things that would be peripheral in a desktop system are built into a notebook. A hard disc drive may be internal or external, or there may be one of each. In principle, there's no difference, so I'm going to treat peripherals as part of the overall computer system.

When I say “computer,” I mean a machine that's flexible in its operation. An MCU, once programmed, is essentially an ASIC or application-specific IC, a chip custom-made to do just one thing. A computer, on the other hand, is designed to run any kind of program, and change programs whenever the user chooses. Way back when the first useful electronic digital computer, the ENIAC, was put into service in 1946, programming flexibility was a basic ingredient of the mix.

The majority of personal computers operate with Microsoft Windows, so that's what I'm going to use as examples. Apple and other systems do all the same things in almost the same ways. Of course, there are also larger mainframe and mini-mainframe computers, but the average tech most often deals with personal computers.

PHYSICAL ORGANIZATION

I'm going to pretend that the desktop computer is still the standard. Laptops and tablets have become so powerful that there's no reason any more for many people to bother with desktops, but desktops still have the greatest capabilities. You also get to put aftermarket stuff into them to increase their performance, while laptops are what they are. I plan to go out and get a new, hot desktop soon, so I'll have one when they become a niche product, which they will. I need one to turn these illustrations into animations for future study guides, and animation takes horsepower.

But, however they're packaged, the functions are the same. The core circuitry in a desktop or laptop computer is contained in a **motherboard**, which is a PC board that's nearly as large as the computer. On the motherboard you will find:

- The **central processing unit** or **CPU**, which does the calculating and also controls data flow etc. This is equivalent to the CPU in a microcontroller but many times faster and more capable. A Pentium would be a representative example.
- **Main memory**, which consists of a little ROM and a lot of RAM, which is opposite from an MCU.
- **Support circuitry** such as video and math coprocessing chips.
- Audio, video and data **ports.**
- **Expansion slots** that allow the “stock” functions of the computer to be augmented. In a laptop, these would have to be peripherals, outside the computer.

Everything else in the computer is connected to the motherboard. This may include:

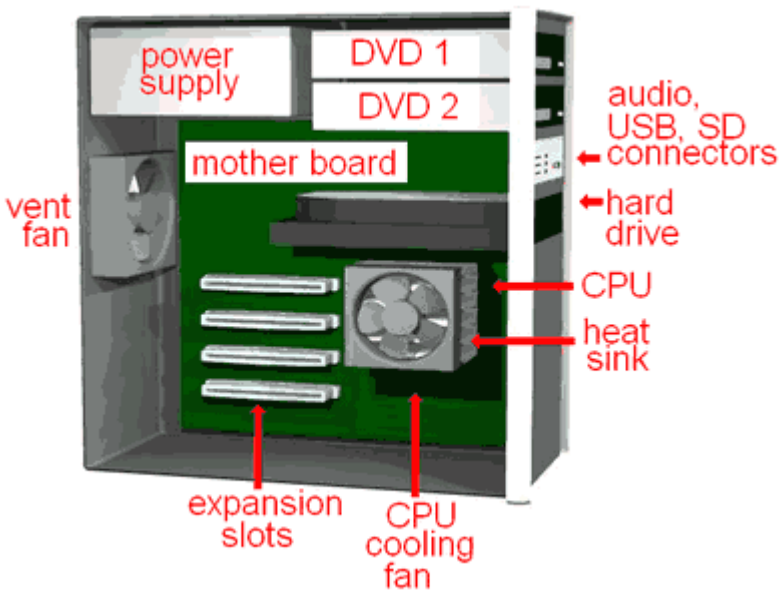
- One or more **hard (magnetic) disc** drives, which typically store 1,000 times more data than there is room for in main memory
- **Optical disc drives**, which use CDs and DVDs
- Special cards which plug into the motherboard's expansion slots. The most common are probably **audio** and **video cards**. A computer comes from the factory with basic audio and video capabilities, but many users – gamers, for example – need advanced audio and video functionality, supplied by the special cards which plug into the motherboard's expansion slots. These cards are called **daughter boards**.

- A power supply which furnishes DC power at several voltages to all the computer's components, including the fans which are needed to remove heat from the CPU and the power supply itself.

Let's look at each of these components.

THE MOTHER BOARD

The motherboard is Grand Central Station; everything in the computer is either located on or connected to it. In a box-type desktop computer, the motherboard is usually placed vertically along one side and it's so large it takes up most of the area from top to bottom and front to back.



In a laptop, notebook or pad, the motherboard is under the keyboard and, again, is almost as large as the computer. A motherboard typically comes from the factory with all the functionality a basic computer needs, only requiring the addition of a hard disc drive, an optical drive, and a power supply. The motherboard is so complete that you can easily buy the components of a desktop computer (motherboard, drives, power supply, case) and assemble one yourself. There are those who feel that if you buy a desktop computer pre-assmebled, you should have your geek credentials revoked. Certainly, rolling your own can save you some bucks.

THE CENTRAL PROCESSING UNIT

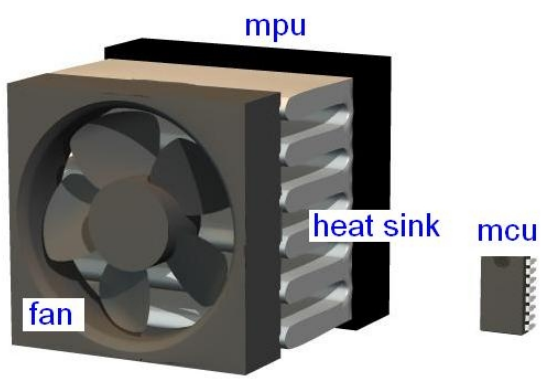
The **central processing unit** or **CPU**, located on the motherboard, is where the computing happens. In the old days, the CPU would take up a whole section of the computer, but now all or most of the computing can be done on a single large chip called an **MPU** or **microprocessor unit**, or just **microprocessor**.

Quick review to keep the letter designations straight in your mind:

- MPU – microprocessor unit, which contains a CPU plus its support circuitry, but not main memory. The microprocessor is the central data processing piece of a computer.
- MCU – microcontroller unit or microcomputer unit, a complete small computer in a single chip. An MCU includes as much memory as needed for its intended control applications. See the previous chapter.
- CPU – central processing unit, where all the data processing happens. MCUs and computers both contain CPUs.

MICROPROCESSORS

A computer's MPU is a lot like an MCU except that it runs at least ten times faster, is much more capable and doesn't contain the main memory. A desktop MPU is typically a brick that's several inches on a side and needs its own heat sink and fan. I've shown it next to an MCU below left, so you can get an idea how it compares. This is some serious number crunching we're talking about.



Programs would run faster if the MPU did contain the main memory, but it's impossible. A computer may have a million times as much memory as an MCU, and there's no way to put that on an MPU chip. So the main memory is contained in chips that are separate from the MPU.

Some CPU functions may delegated to other chips. There's always a balancing act when designing a computer: if the MPU does everything by itself, performance takes a hit; if some functions are offloaded to support chips so the MPU can run faster, your bank account takes a hit. Gaming video in particular hogs a lot of the MPU's capability. So, the hotter the computer, the more functions you'll find being handled by other chips related to the CPU, or even separate PC boards, and the higher the cost.

Modern computers typically “think” in 64-bit words. A 64-bit number is, in decimal, about 1.8×10^{19}, or about eighteen quintillion. There's seldom any need to process a number that large. One reason for such a large word is to process other things – video images, for example – which involve a lot of data that needs to be handled very quickly.

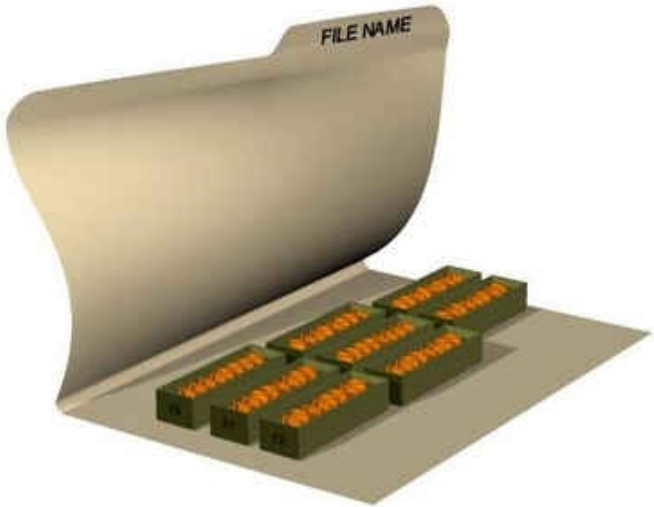
But the main reason we've gone to 64-bit processing is that a 32-bit processor can only address about 4GB of memory. At one time, 4GB was a large hard drive. Today, it's a small USB drive. A 64-bit processor could handle 4 billion 4GB drives.

MULTIPLE CORES

It's almost a law of nature that computers will keep getting more powerful. The standard ways of doing that have been to increase speed and increase word length, but we've reached the point where it's hard to make CPUs run faster, and to date it hasn't made much sense to go beyond 64-bit words. So, to keep pushing the boundaries, computer designers are providing them with multiple **cores**, several separate MPUs' worth of computing hardware. The reason that didn't happen decades ago is that it's trickier than you'd think to divide up the computing tasks. Now that the gurus have finally figured out how to do it – frankly, now that we're reaching roadblocks in terms of speed and size, and have no other way forward – we can expect newer software to be written for multiple cores as a matter of course.

MEMORY

In an MCU, which is programmed to do only one thing, data is just data in memory. In a computer, because of all the different programs and types of data, the data is organized into **files**, so named because they resemble folders in a file cabinet. Each file contains however many bytes are necessary to hold a related bunch of data. As with an MPU, memory is organized into **registers** of one byte each. Each register has an **address**.



Though the CPU may think in 64- bit words, memory is still usually organized in 8-bit bytes. Where larger words need to be stored – for example, high-quality video requires 24 or more bits of information at every point on the screen – the CPU breaks large words into bytes and stores them in several registers, then reassembles them later.

Common types of memory:

- **Main memory**, contained in semiconductor chips. This is equivalent to the RAM and ROM in an MCU, except that there's a million times more of it, and it's mostly RAM.
- The **cache**, a comparatively small amount of semiconductor memory contained in the CPU.
- The **hard drive**, where data is stored magnetically.
- **CD** and **DVD drives**, where data is stored optically.
- **USB / thumb drives** and **micro-SD** chips that use flash memory.

ROM

Unlike an MCU, there is very little ROM in a computer MPU because there's very little firmware. The ground-level program that runs the computer, called the **operating system** or **OS**, is not permanent as it is in an MCU. It's stored on a hard disc, which we'll get to in a bit.

In Windows-type computers, the program stored in ROM has traditionally been called the **BIOS** for **basic input-output system**. When power is first turned on, the BIOS checks to see what hardware is present – keyboard, monitor, disc drives etc. Then it goes to the hard disc and finds the program called the **boot loader**. This is a program that causes the OS to be loaded into RAM. Every time the computer is turned on, the whole of Windows or whatever operating system the computer uses is read off the disc and into the computer's memory. This is called **booting up**, which is short for **bootstrapping**; the computer has just enough intelligence in ROM to find and load the much greater intelligence in the OS.

It turns out that BIOS is vulnerable to malicious virus programs, and it has other limitations inherited from now-obsolete hardware. A more secure boot program called **UEFI** for **unified extensible firmware interface** has recently started to supersede BIOS.

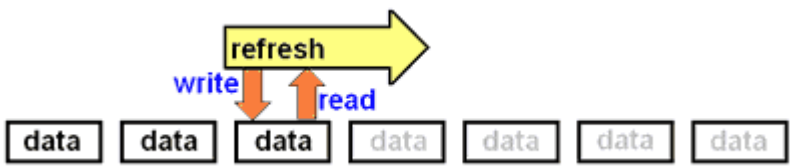
RAM

The fundamental difference between an MCU and a computer is that an MCU has a defined function, while a computer's functions are defined and redefined by the user every time a new app is opened. For this reason, the semiconductor **main memory** in a computer is almost all RAM, not only for data but for the various programs the computer is working with and the operating system itself. Last time I checked, in January 2012, a typical desktop computer has up to 8GB, 8×10^9 bytes, of RAM.

In chapter 5, we saw that RAM is volatile, meaning that it forgets its data when power is turned off. Most of the RAM in a computer could be called “ultra-volatile”; it forgets its data even when power is on, unless it is reminded every few milliseconds. This kind of RAM is called **dynamic** RAM, as opposed to **static** RAM like that found in most MCUs, which remembers its data as long as power is turned on.

Each binary bit in dynamic RAM involves a tiny capacitor where a charge is stored. The capacitor is so small that the charge leaks off almost immediately, which is why dynamic RAM needs to be reminded of its data so often. This is called **refreshing** the memory.

Every few milliseconds the OS comes around, reads the decaying data in a register, then rewrites it back into the register at full strength, like a juggler who keeps catching balls before they hit the ground and tossing them back into the air.



Dynamic RAM is used because it's smaller, simpler and cheaper than static RAM. Each bit consists of a capacitor and one transistor, as opposed to half a dozen transistors for a static RAM bit. Static RAM is better in theory, but you couldn't fit enough of it into a laptop or afford to buy it.

ADDRESS, DATA and CONTROL BUSES

When we were talking about microcomputers, we looked at Von Neumann architecture, which uses one bus, and Harvard architecture, which uses two. There was no need for a separate control bus because everything was together in one IC.

In a computer, there are many parts outside the CPU and in a desktop system outside the computer itself. To keep track of things, and to prevent a logjam on a single bus from slowing things down, a CPU typically has more than two buses. It's dangerous to generalize, as different CPUs have different designs, but buses can usually be grouped into three types:

- a **control bus**, which carries commands
- a **data bus**, which routes data
- an **address bus**, which identifies the address of the memory location or other device that the CPU wants to read from or write to

BUS SPEED

The dirty little secret of computers is that computing speed is largely determined not by the clock frequency, which may be several GHz, but by the bus frequency and RAM speed, which are much lower and almost never advertised.

A large percentage of computing time is taken up waiting for data to travel back and forth between the CPU and main memory. In current computers, the main bus between the CPU and memory (called the front side bus by Intel and the hyper-transport bus by AMD) typically runs at 800MHz. Main memory is usually faster than this, and the CPU is usually at least twice as fast. The bus does not affect how fast the CPU crunches numbers it already has at a particular moment, but it slows things down whenever the CPU wants to read or write numbers to memory. Depending on the application, bus speed may be more important than CPU speed.

THE CACHE

To ease this problem, the MPU is built with several MB of RAM, called the **cache**, to hold the most immediate data right in the MPU.

The cache is essentially a mega-accumulator. Another illustration of how a computer compares to an MCU: an MCU may have from 1kB to 20kB of memory, total, while a computer's MPU has a thousand times as much in the cache alone.

The speed bottleneck is becoming such an issue that CPU makers are looking at splitting main RAM and putting part of it inside the CPU. By 2015, we should see CPUs with several gigabytes of RAM. This will greatly speed up program execution, even if the basic machine cycle doesn't get any faster.

THE MCUs

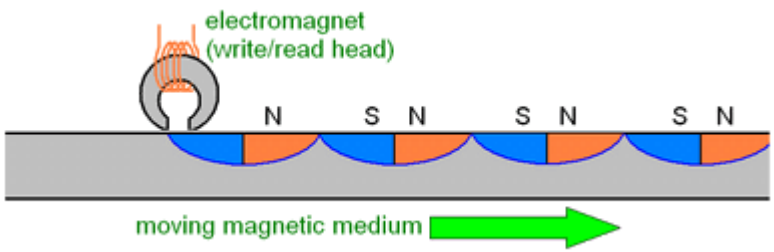
MCUs? Wasn't this supposed to be a computer discussion, and isn't a computer run by a central microprocessor? Yes and yes, but in a desktop system at least, there are also a dozen or so MCUs scattered around. Everything that has an identity of its own – monitor, keyboard, mouse, printer, disc drive, everything but the speakers – needs embedded smarts to operate, and to interface with the main computer. In a computer system, an embedded MCU is like a little machine that manages a small part of the overall job, like the person who empties wastebaskets in a large office.

THE HARD DRIVE

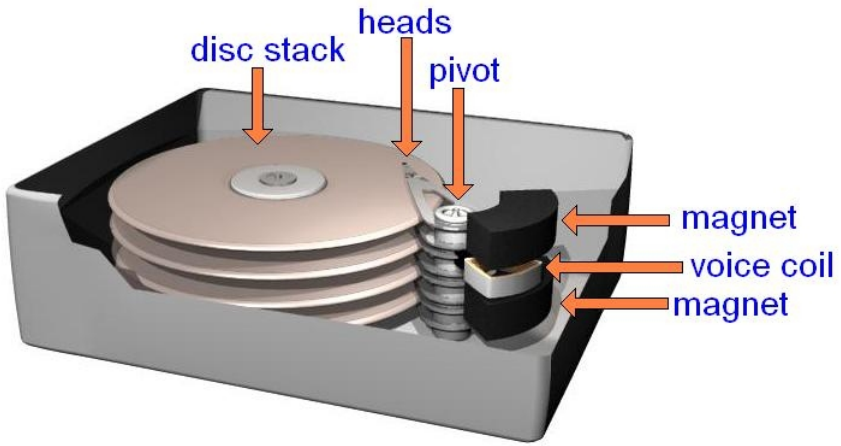
In EM2, you learned that current through a wire coil creates a field, and a field moving through or past a coil creates a voltage. Way back in 1898, a Danish guy named Valdemar Poulsen demonstrated that sound could be recorded by using an electromagnet to magnetize a moving steel wire.

When the wire was moved past the electromagnet again, the magnetization induced a voltage in the electromagnet and sound was played back. This was the principle behind audio and video tape and cassette recorders, and is the principle behind the hard drive.

A **hard disc** (or disk) is a spinning disc coated with ferromagnetic material. As it spins, an electromagnetic **read/write head** near the surface of the disc creates submicroscopic areas of North and South magnetization to represent 1s and 0s. When those tiny magnets come around again, a voltage is magnetically induced in the head which can be turned back into 1s and 0s.



There is room for many parallel rows of data, called **tracks**. To read or write a track, the head is physically moved across the disc. This was once done with motors and spiral-threaded rods, but today the head is mounted on a pivot and positioned by a magnetic apparatus called a **voice coil**. In a loudspeaker, the "voice coil" is what pushes the cone back and forth to make sound. That idea was borrowed for hard discs, and the name came along for the ride. In the illustration below, the function of the voice coil may seem confusing. The magnetic fields are arranged so as to push sideways on the voice coil, pivoting the head stack.



A **hard drive** is a stack of hard discs with a stack of read/write heads, all on the same pivot, all positioned by the same voice coil. The "hard" designation is unnecessary today, as all magnetic memory discs are hard. But at one time, computers also used **floppy discs**, where the magnetic material was coated on flexible discs of plastic film. Flash memory has made floppy discs obsolete, but the name for the other kind, "hard drive," has stuck.

The discs spin continuously whenever the computer is on; the heads “float” above the discs on a thin cushion of air, so there's no surface wear. The air cushion is so thin that a speck of dust would jam itself between head and disc, gouging the disc. For this reason, hard drive enclosures are sealed. The illustration on the previous page is a cutaway view.

Hard disc drives have traditionally stored the most data in the least space. Today's hard drives often have a terabyte of storage, 10^{12} bytes, 1,000 times more than the computer's main RAM. In addition to data, the hard drive stores the computer's operating system and any programs that have been installed in the computer. The OS is copied from the hard drive and into RAM when the computer is booted up; other programs are copied into RAM when the user calls for them.

A hard drive is, in effect, nonvolatile RAM; it is random-access and can be written to and read from like RAM. Its virtues are its nonvolatility and its great data-storage ability; its drawback is its slowness. The time it takes to access a track is the time to move the heads and rotate the disc into the right positions. Rotating the discs and moving the heads takes time – only milliseconds, but that's orders of magnitude slower than solid state RAM.

If execution of a program requires more RAM than is available, the OS will use the disc drive for overflow. This slows down execution, which is one reason many users load up their computers with as much RAM as possible.

ERROR CORRECTION

When we talk about magnetic drives, we have to deal with something we haven't dealt with before. Data in an MCU or a computer's solid state memory very seldom gets corrupted. But with magnetic media, we're talking about a mechanical system where the occasional bit gets lost. I'm not saying it *might* happen; it *does* happen, all the time. Discs have defects; that's just a fact of life. So data gets messed up.

In many applications, such as bank records, that's unacceptable. It's also unacceptable in a program, because a corrupted byte runs the program off the track and it crashes. Obviously, if we want to make magnetic media work we have to develop ways to automatically detect and correct errors. I'm going to show you a couple of the very most basic methods used in early computers.

The simplest error correction is to add a bit to each byte that tells you whether the sum of the bits is supposed to be even or odd.

For example,

00010110

is odd, and by this I don't mean the number itself is odd – it's 22_{10}, which is even – I mean that there are three 1s, and three is odd. So, let's say we decide that “odd” represents 1 and “even” represents 0. We add a **parity bit**, in this case a 1, to the end of the byte:

000101101

Now, suppose we're reading this number back from a disc drive and it shows up as:

010101101

The number of 1s is even, but the parity bit says it should be odd. This tells the computer the byte has gone rotten. The error has been detected, but the computer can't fix it because it doesn't know which bit is wrong.

The next level of error detection is to take eight bytes and add vertical parity bits:

000101101
100101001
101101000
100011100
000110100
010110001
110000111
011101110
01101110

Both vertically and horizontally, there's one byte where the parity bit is wrong, shown in red. That lets the computer pinpoint the error; a “0” in the fifth word down.

Once the error has been located, it can be corrected. Since there are only two states in binary, if a 0 is wrong, then a 1 must be right. Duh.

As error correction gets more complex, the amount of data used for correction can become as large as the original amount of data itself. And then there are such techniques as breaking the data up and spreading it around, so a defect in the media can't wipe out a whole section of data, and recording more than one copy of the data, which is known as **redundancy**.

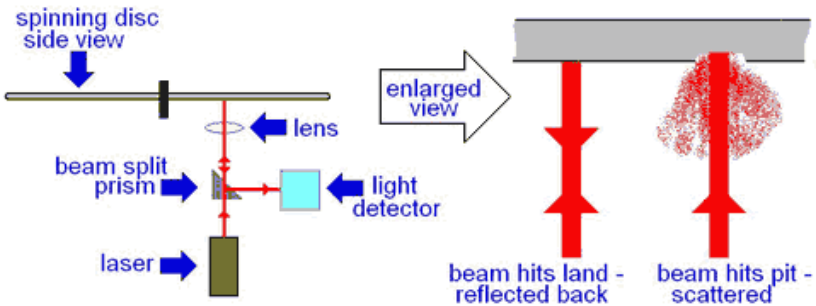
You'd think these measures would reduce the amount of data we can store, but they don't. The more densely we pack data, the more data will be corrupted by a defect. But the more powerful our error correction, the more data corruption we can afford. If elaborate error correction means we can pack data more densely, we gain more than we lose. Hard drives keep growing in capacity not so much because they actually keep getting bigger but because our ability to correct errors keeps getting better, permitting us to pack more data in the same space. The terabyte drives of today aren't much different from the 250GB drives of a few years ago, but now we have more powerful error correction.

I mentioned that there are MCUs sprinkled around the computer in addition to the main CPU. The disc drive is one example. Managing everything that goes on – spinning the discs at the proper speed, keeping track of where they are in their rotation, moving the heads, buffering data in and out – requires enough digital smarts that there's always at least one MCU in a disc drive. This keeps the main computer's MPU from having to think about it. It also means a given computer can work with a variety of disc drives from different manufacturers.

One of the things the drive's MCU does is to automatically mark defective disc sectors. When reading data off a disc, if the error correction software identifies a disc sector as having an especially high error rate, the MCU will make note of its location, copy the corrected data to a different sector, and then take the defective sector out of service before it gets bad enough to really screw things up. This all happens automatically, within the drive, without the involvement of the computer's CPU or the user.

OPTICAL DISC DRIVES

A hard drive is based on magnetism; CD and DVD memory is based on light. A CD or DVD is a sandwich, and one layer is metal film. When data is written to the disc, a laser burns microscopic **pits** in the metal layer. To read the data out again, a lower-power laser (or the same one, operated at a lower power) is used to illuminate the pits. The “land” area between the pits is shiny, and reflects light back toward the source; the pits, on the other hand, scatter the light. This means the amount of reflected light will differ, and that can be translated to 1s and 0s.



Older CDs and DVDs could only be written to once. All the data that will be on the disc is placed in a RAM file in a computer, and when that file is complete, the data is transferred to the disc. This is called **burning** a disc, an apt term since the writing laser literally burns pits into the aluminum layer. Once burned, a CD or DVD is essentially a ROM because it can only be read from, not written to; thus the terms **CD-ROM** and **DVD-ROM**. The main difference between a CD and a DVD is capacity; a CD-ROM stores upwards of 1GB of data and a DVD stores 5-10GB. Blu-ray discs can store 50GB, and that capacity will increase. It remains to be seen whether this will mean much to computer users, who are moving away from optical discs entirely in favor of downloading straight from web to hard drive.

Some CDs and DVDs can be rewritten. Instead of an aluminum layer, a rewritable disc has a layer of crystal material that can change state. If the laser hits it with full power, it cools into one state; if a lower power, it cools into the other state, and the state can be switched back and forth. The two states have different reflectivity so a low-power laser can read 1s and 0s from them as with a regular CD or DVD.

There's a misconception that CD and DVD ROMs are immune to water and chemical damage because the aluminum layer is sealed in plastic. In reality, water can soak through the plastic. If a CD or DVD is left out in the rain for a month, the aluminum layer will be ruined. But if kept in a dry, temperate environment, CDs and DVDs last longer than almost any other data storage method. For critical applications, special archival CDs and DVDs are available. And when archiving data, the fact that a non-rewritable CD or DVD can only be written once is an advantage; the data cannot be accidentally overwritten.

EXTERNAL MEMORY

If the computer's internal memory isn't big enough, more memory can be added in the form of an external hard drive or in the form of **thumb** and **micro SD drives**.

Strictly speaking, the latter aren't drives. A drive is something with a motor in it, like the motor that spins the hard drive's discs. However, to most people the word “drive” means a gizmo with a large chunk of memory, and that's what thumb and SD units are.

Thumb and SD drives use **flash** memory. To review from chapter 5, flash memory is like dynamic RAM in that it stores a 1 or 0 as an electrical charge. In this case though, the charge is stored on a MOSFET gate that “floats” – it isn't connected to anything.

The gate is insulated from everything around it, and with today's technology the insulation can be made so good that the charge lasts for decades. The gate is charged or discharged by putting a high enough voltage across the insulation that electrons tunnel through.

External flash memories aren't big enough to add significantly to the computer's overall memory. Their purpose is to transport files to other computers. This is what floppy discs were for, but the smallest thumb drive has 1,000 times the capacity so nobody uses floppies any more. At this writing, a large thumb drive is 32GB. Terabyte thumb drives are becoming available (for a lot of money). That's enough memory to store a full, uncompressed, high-definition feature film.

Flash memory is not ideal for archiving data because the charge on the MOSFET gates eventually dissipates. Also, flash drives wear out; after several thousand writes to a particular register, it becomes unreliable. (There is deluxe flash memory that will take tens of thousands of writes, but it isn't found in thumb and SD drives.)

THE SOLID STATE DRIVE

A solid state drive or **SSD**, also called an **electronic disc**, is a solid state static RAM memory so large it begins to compare to a hard drive. Currently, several hundred gigabytes is a typical size, which was the size of a hard drive six or seven years ago. The advantages are faster read/write times and freedom from the fragility and reliability problems associated with a mechanical hard drive.

An SSD can be combined with a regular hard drive to provide a hard drive's large data storage capacity along with most of an SSD drive's speed.

DAUGHTER BOARDS

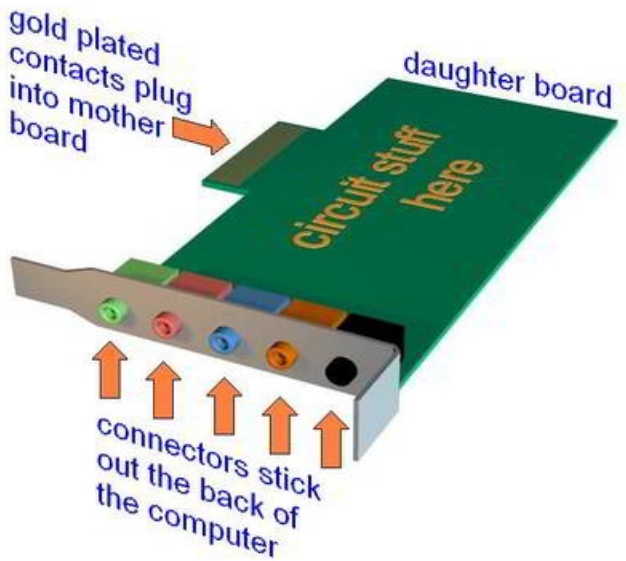
A computer's main PC board isn't called the mother board just because of its size and central importance. Often, smaller printed circuit boards, called **daughter boards**, are plugged into the main board via expansion slots.

Digression: any time a smaller board attaches to a larger board, usually at right angles, the larger board is a mother board and the smaller board is a daughter board. I suppose if the smaller board had a still smaller board attached to it, that would be a granddaughter board. This technique, and the names, are not limited to computers.

In a computer, daughter boards are used for two reasons:

- To expand the capabilities of the computer, so it can do things it couldn't do in its factory-fresh, “stock” condition (that's why the slots are called expansion slots), and
- To speed up the computer's normal functions by off-loading some tasks from the CPU.

Daughter boards for desktops are designed with connectors on two sides. On one side are the gold-plated “fingers” that go into the motherboard's expansion slots. On the other are connectors that appear at the back of the computer, where the user can make direct connections.


gold plated contacts plug into mother board
daughter board
circuit stuff here
connectors stick out the back of the computer

Probably the two most common examples are video and audio cards. The average computer has video and audio capabilities on the motherboard, but they're limited. For example, my desktop computer, which is old enough that it will soon qualify for Medicare, came with a VGA connector for the monitor, and microphone in, line in and speaker out connectors for audio. But I never use those connectors. I used two of the expansion slots to add a video card with dual outputs – one DVI – and an audio card with 7.1 channel surround sound. These cards come with their own connectors, and those are the ones I use.

High-definition, high-speed video takes up an enormous percentage of a CPU's computing power. Just transmitting high-definition video takes up hundreds of megahertz of bandwidth. But that video must first be generated, which is a much larger task. There's no way a CPU can do 60fps high-def video along with everything else. So, if you want good video performance, you have to plug a video card into an expansion slot.

What really drove the advancement of video was gaming, where the image is computed in real time, 60 times a second. The 3D graphics you see in these study guides, like the illustration at left, would have taken forever to produce without a dedicated video card in my computer.

Audio is much less computation-intensive; the main reason for installing an audio card is to get more channels. The basic computer audio connections are:

- Microphone input: pink. High impedance unbalanced, with a few volts of DC to power condenser mics
- Line input: blue. High impedance unbalanced
- Line/headphone output: green. Unbalanced, designed to drive a 32Ω or higher load.

Connection is via 1/8” or 3.5mm plugs and jacks. The tip is left channel, the ring is right channel, and the sleeve is ground.

The sound card may support additional channels, all high impedance, line level, unbalanced:

- Center and subwoofer output: orange.
- Rear surround left & right: black.
- Middle surround left & right: gray.

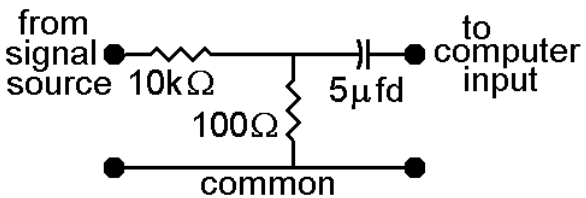
There's also one non-audio 1.8” jack:

- Midi/joystick port: gold.

The other reason for installing an audio card is to get better sound. Audio from the motherboard is sometimes marginal. In my old desktop, some of the digital waveforms leak into the audio, creating beeps and buzzes that drive me nuts. I'm sorry to report that newer computers aren't much better. Plugging an audio card into an expansion slot gets you very good quality, but the best audio is obtained from an external card connected to the computer via USB.

Speaking of USB audio, there's a good reason for the current proliferation of USB-enabled microphones. The pink microphone input on an audio card is often crap – noisy, poor frequency response, easily overloaded. It's intended for a boom microphone attached to a headset. It's not intended for studio quality recording.

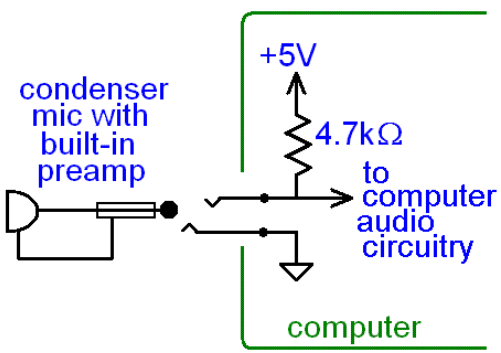
In many computers, there is no blue line-level input. Here's the circuit again that I first showed in EM3, an adapter that changes line level to mic level and blocks the DC on the mic jack. In some computers, the DC can be disabled and the sensitivity of the mic jack reduced via software commands, turning the pink connector into a virtual blue connector.



Still, best sound is obtained by converting audio to digital externally and feeding it in on a USB port.

MICROPHONES

The pink microphone input is intended for one type of mic and one only: a medium-to-high impedance, unbalanced, condenser mic. Any other kind of mic requires an adapter, not only to change one type of connector to another but to manage electrical compatibility.



A condenser mic requires a DC power supply. In the kind of mic intended for a computer, DC is supplied via a resistor, typically about 5k ohms. The resistor keeps the power source, which is very low impedance, from shorting out the audio.

The pink mic input is not designed for extremely high quality. In particular, its noise floor is far above what could be achieved. The reason this doesn't cause a problem is that condenser mics are sensitive (high output for a particular sound level), and because the mic is usually used placed close to the user's mouth, resulting in a hot signal. Where better quality is desired, an external preamplifier with a USB output should be used.

LOUDSPEAKERS

Loudspeakers come in two flavors: **passive** and **powered**. The difference is that powered speakers have power amplifiers built into them.

Most computers have a single green output jack that serves both as a line output and a headphone output. As discussed in the EM3 study guide, audio equipment does not use impedance matching, except in the telephone industry. An audio output is designed to work into any load

from a specified minimum impedance up through infinity (open circuit). Since most headphones nowadays are 32Ω per side, the green jack is designed for a 32Ω load. When used as a line output, working into a light 10kΩ load, it kicks off its shoes and takes it easy.

Driving a speaker is another matter. Speakers are usually 4 or 8 ohms, and it takes more power to drive them than is available from the green jack. For this reason, most speakers intended for use with computers are powered. The most common power source is a 117VAC transformer in a small box with two prongs sticking out, commonly referred to as a **wall wart**. I'm fond of that expression because my old boss hates it. He thinks it's undignified.

It's hard to get good low frequency response out of the kind of tiny speakers you want on your desk, so tricks have been developed to make you think you're hearing more bass than you are. The one non-tricky, legitimate way to handle bass is to strain it off from the desk speakers and feed it through a **subwoofer** by your feet. For every low-frequency octave you remove from the desk speakers, the required enclosure volume for a given level of performance goes down by a factor of eight – half the width, half the height, half the depth.

People doing serious, recording-studio work know better than to use the built-in audio capabilities of the computer; at a minimum they add a sound card, preferably an external one. They also use big speakers with a very wide response, down to 30Hz, or expensive headphones. They're not necessarily creating program material that goes down that low, but if there happens to be anything down there, they should have a way of hearing it. You'd be surprised how much low frequency garbage gets into audio files because the producers can't hear it on their monitors and don't know it's there. I was just listening to a 1950s major-label record of a big-name jazz singer and she's popping her mic all over the place.

PORTS AND DIGITAL COMMUNICATION

With the exception of analog audio and video, every signal that goes in and out of a computer is typically in digital form. A digital connection to a computer is, like a data connection to an MCU, called a **port**.

PARALLEL PORTS

A parallel port has eight wires to send a full byte in or out at once, plus other wires for other stuff. I'm not going to bother showing you the pinout, because many different connectors have been used and because parallel ports are obsolete. You find them on old computers and old machines.

SERIAL PORTS

A serial port sends data one bit at a time. In theory, this takes eight times as long as a parallel port, but it has proved easier to speed up transmission by increasing bit rate than by using more wires. When you must connect to an old parallel port, you can get an adapter.

We'll take a brief look at a few of the most-used communication standards beginning on the next page.

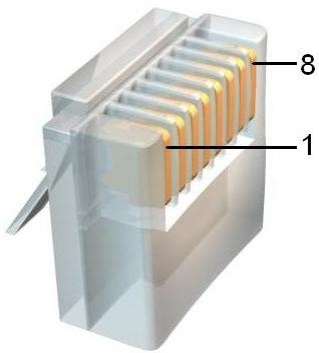
RS-232

The most-used serial data format for many years has been the **RS-232** standard, which has the following characteristics:

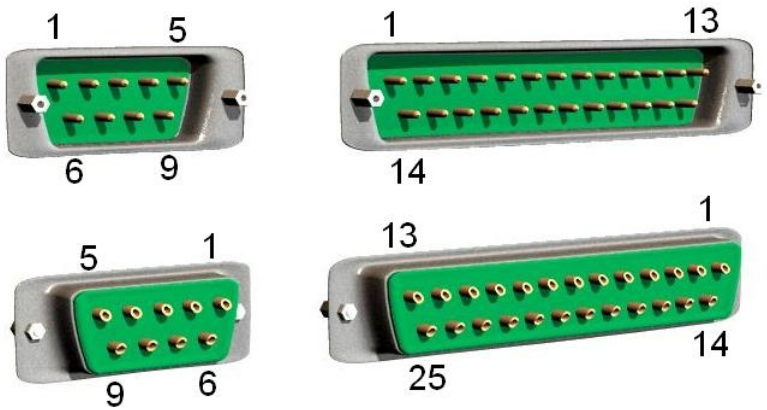
- Voltages: “0” is positive by up to 25V, “1” is negative by up to -25V. These high voltages are a holdover from the days of clackety-clack teletypes once used by newspapers. Today, 5V to 12V is more common. The higher the voltage, the more resistant the system is to data corruption by noise.
- Data is sent on a single wire referenced to a ground wire; that is, it's unbalanced. This can cause major problems when the link connects two distant pieces of equipment that are each grounded locally. (A detailed discussion of grounding problems can be found in the EM3 study guide.)
- Word size: sometimes 5 or 6 bits, or 7-bit ASCII, but in modern equipment almost always 8 bits. Words are usually sent least significant bit first.
- Parity bit: a parity bit is added to every word. The receiving unit can cross-correlate to correct many errors, but if an error cannot be corrected, the receiving unit automatically requests a retransmission. (This isn't automatically true; many RS-232 links are simplified and don't have handshaking capability. But the standard does provide for handshaking if you want it.)
- Start and stop bits: a start and a stop bit is added to every word. These are bits that are always the same, so the receiving unit can tell them from data bits and divide the serial bitstream into words.

- Number of wires: the RS-232 standard specifies 20 different signals, the majority of which are seldom used. RS-232 can even be sent over a 3.5mm headphone-style plug or a subminiature 2.5mm one, which has at most three connections, if you can do without the handshaking.
- Connectors: the most common connectors are **DB25**, **"DB9"** and **TIA-561**, which is also known as an **8P8C** or **RJ45** modular connector. (The designation DB9 is incorrect, but so many people use it that it's become the de facto standard.)

8P8C



DB9 DB25



RS-232 was originally developed for a maximum data rate of 20kbits/second. This has gradually been increased to the point where some RS-232 systems operate at 1.5Mbit/second. Officially, maximum cable length is 50 feet, but this depends on the cable. The real bottom line is that cable capacitance must be 2,500 pF or less. Low-capacitance cable can be used to achieve longer distances. Maximum cable length also depends on data rate. At 2kbits/second, which is more than fast enough for many machine control applications, RS-232 can work over distances of thousands of feet.

RS-232 is designed as a **one-to-one** link; if a computer operates several external devices, it must have several RS-232 ports, or an external device that converts one port to several.

The chart on the next page shows the connections; the ones that really matter are **transmit data**, abbreviated **TxD**; **receive data**, abbreviated **RxD**, and **ground**.

Name	Typical purpose	DB25	DB9	8P8C/RJ45
Data Terminal Ready	OOB control signal: Tells DCE that DTE is ready to be connected.	20	4	3
Data Carrier Detect	OOB control signal: Tells DTE that DCE is connected to telephone line.	8	1	2
Data Set Ready	OOB control signal: Tells DTE that DCE is ready to receive commands or data.	6	6	1
Ring Indicator	OOB control signal: Tells DTE that DCE has detected a ring signal on the telephone line.	22	9	
Request To Send	OOB control signal: Tells DCE to prepare to accept data from DTE.	4	7	8
Clear To Send	OOB control signal: Acknowledges RTS and allows DTE to transmit.	5	8	7
Transmitted Data	Data from computer to peripheral	2	3	6
Received Data	Data data from peripheral to computer	3	2	5
Common Ground		7	5	4
Protective Ground		1	—	—

2, 3 AND 5-WIRE RS-232

Where the full control capability is not needed, RS-232 can be implemented using a send wire, a receive wire, and a ground wire. If the information flow is one-way, two wires will do. If handshaking is required, a 5-wire system is used.

The RS-232 standard is becoming obsolete in computers, replaced by the USB standard. However, it's still widely used in machine control applications because it's sturdy and because data can travel over long distances. At least one computer manufacturer that had eliminated the RS-232 connector has brought it back for that reason.

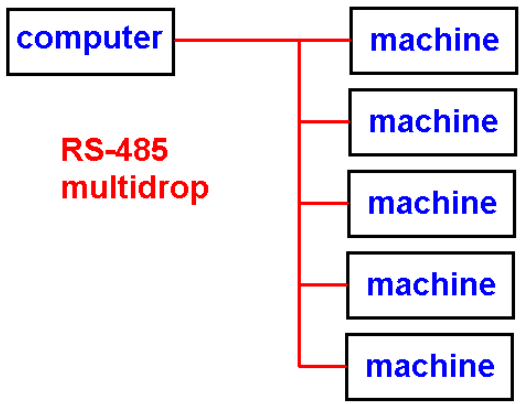
SIMPLEX, FULL DUPLEX, HALF DUPLEX

Three terms you should know:

- Simplex: one-way transmission.
- Full Duplex: two-way simultaneous transmission. This usually requires separate send and receive wires.
- Half Duplex: two-way transmission, but only one way at a time. Each unit takes turns transmitting and receiving. Walkie-talkies are half duplex.

Full-blown RS-232, with a DB9 or DB25 connector, is full duplex.

RS-485



RS-485 is widely used with general-purpose industrial controllers. It's a 3-wire system, where data is sent over a balanced twisted pair that is referenced to ground; i.e. not floating. It's designed so the single pair can be run all over a building, connecting to a unit called a **node** or **station** wherever necessary. In other words, it's a big data bus, meaning that the nodes must take turns using the bus in the same way as the bus inside a computer. A single data line (or pair) that goes all over a building and connects to many nodes is referred to as a **multidrop** system.

The data wires must be provided with default bias voltages so they don't float when no station is using the bus. This is to prevent the line from picking up noise, which the stations might incorrectly interpret as data. The pair should also be terminated to prevent signal reflections of the type we discussed in EM3. Usually, both functions are performed by a resistive voltage divider.

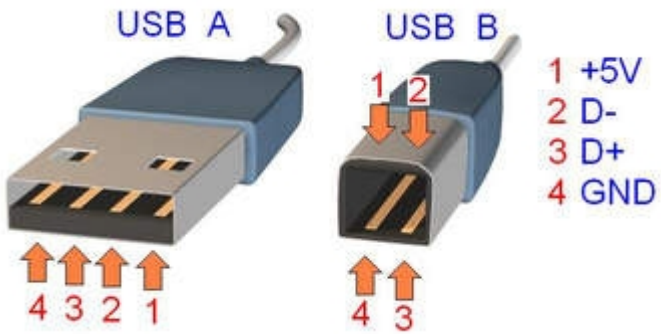
An RS-485 system has a master control unit and a number of remote stations. Every station connects to the bus via tri-state circuitry. The control unit polls each station in turn, asking, “do you have data to send?” Its output then goes into hi-Z state. If the station has data to send, it sends. If not, after a short wait, the control unit goes back into low-Z mode and polls the next station.

Early RS-485 equipment had no protection against ground-potential differences and used CMOS chips directly connected to the bus, a sure recipe for disaster. Very often, a zapped station would fail to go into high-Z mode, hanging onto the line. It would continue to work fine, but undamaged stations around it would malfunction because the zapped unit would short out their data. This problem was solved with the introduction of hybrid ICs that incorporate isolating circuitry. Modern RS-485 equipment is fully isolated at the connection point to each node and has no ground-loop problem.

USB

For short-distance data transfer, the **universal serial bus** or **USB** has been gradually replacing RS-232 since its introduction in the mid-1990s. Physically, it's much simpler than a full-featured RS-232, as it has only four wires: +5VDC, +data, -data, ground.

All the control functions are multiplexed into the data, and the data wires are bidirectional, sending or receiving data by turns, meaning they're half duplex. The data lines form a balanced pair, both to avoid picking up interference and to avoid radiating it. Thanks to the 5V pin, small peripherals can be powered by the USB port instead of requiring their own power supplies.

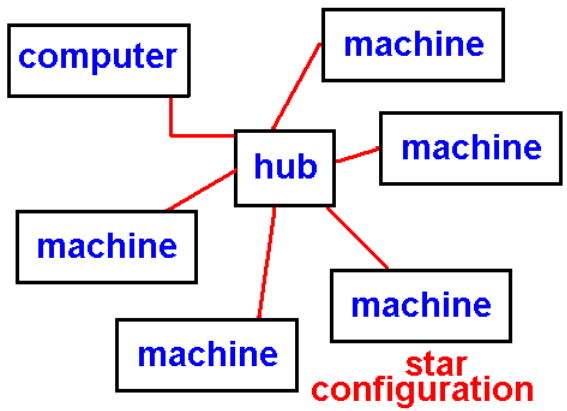


USB connectors come in “A” and “B” versions. “A” versions are found in devices which supply +5V power. “B” versions were originally designed for devices which use power. Because of these very different, incompatible connectors, the user cannot accidentally connect two 5V power sources together, which could lead to the power supplies fighting each other. In the illustration, pins 1 and 2 of the USB-B connector are hidden under the top. “B” connectors are getting rare, as everybody wants every USB connector to plug into any other USB connector. The power supply circuits in equipment have been modified so they don't fight.

Micro-USB connectors come in several shapes. These include a fifth pin which electrically identifies an “A” or “B” connector.

The USB system would have replaced RS-232 completely except for one problem: it's limited to a length of about 15 feet. However, there are extenders available.

A USB port can support multiple peripheral devices using a **USB hub**, a box that expands a single port into multiple ports. This is often called a **star** configuration, which can be thought of as several 1-1 connections. The hub is powered by the computer's USB +5V supply. When a USB device is connected to a computer, the computer assigns it a device name, and includes that name in any communication intended for that device.

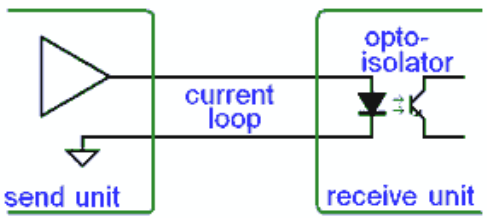


This is equivalent to the way an MPU distributes data over a common data bus, and the reason a single pair of data wires can be shared by many USB devices.

USB devices can be plugged in, unplugged or changed at any time, even while the computer is running. This is called **hot swapping**. At most, you have to tell the computer that you're about to unplug a device, so it won't be in the middle of reading or writing when you unplug it.

MIDI

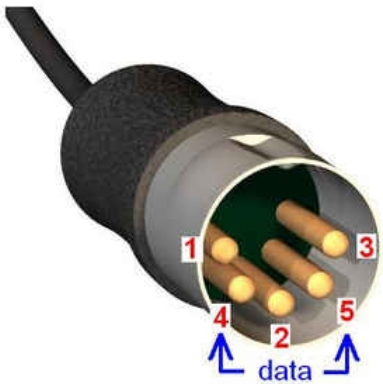
MIDI stands for **musical industry digital interface**. It was designed to allow musicians to plug multiple keyboards (piano-type) together and play them all from one keyboard.



The standard has since been borrowed for other uses such as stage lighting. MIDI is now routinely controlled by computers, which is why MIDI qualifies as a computer interface standard.

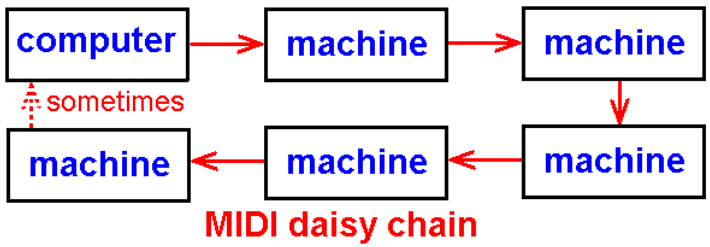
MIDI is a balanced pair with an opto-isolator on the receiving end. This is done to eliminate the ground loop that the MIDI cable would otherwise have caused.

MIDI is one of the few communication systems that properly deal with the ground loop problem. The signal format is a **current loop**; information is transmitted not as changes in voltage, but changes in current.



At the keyboard or sending-device end, classic MIDI uses a standard 5-pin European **DIN connector**. Pins 4 and 5 were chosen because they're the pins least often used for other things. If a computer is part of the system, the connector is not likely to be a DIN connector today. Special cables and adapters are available to connect from DIN to computer. Today, many MIDI devices also have USB connectors so you don't have to bother with the DIN connector, which is hardly used anywhere else.

Multiple piano-type keyboards are handled by plugging the second keyboard into the first, the third into the second and so on. This is called a **daisy chain** configuration. At each device the data is received via an opto-isolator and sent on to the next device, so ground loop protection is maintained. This optically-isolated, ground loop proof, daisy-chain feature is what has made MIDI attractive for non-musical uses.



MIDI is normally one-way or simplex. However, the end of the daisy chain is sometimes connected back to the first device, in which case data can be sent from any device to any or all other devices.

Ethernet

Ethernet is a standard used to create a **LAN** or **local area net**, a wired network that connects computers and related devices within a building. For example, in a large bank, there may be a central data-storage computer called a **server**, many individual computers, and a number of high-speed laser printers. The LAN connects them together and to the internet and permits data to be sent back and forth. Ethernet is a multidrop system and every device is a node.

Ethernet data is sent in packets which, in Ethernet, are called **frames**. A frame consists of:

- A preamble, which synchronizes the system
- A start of frame indicator
- Which device is supposed to receive
- Which device is sending
- Details about the data
- Data (finally!), ^{64}-1522 bytes
- Error detection & correction (parity etc)
- Space before the next frame

The reason the receiving device must be specified is that, unlike RS232 where each like is a hardwired 1-to-1 data path, or an RS485 where a master unit polls each node in turn, in Ethernet any device can originate a communication at any time and everything is sent everywhere. The sending device must also be specified because the receiving device checks every single frame and returns either an “ok” message back to the sending device, telling it to send the next frame, or a “not ok” message, telling it to resend the last frame.

In a frame with only 64 bytes of data, the stuff around the data exceeds the amount of data itself. One of the reasons for all of that is that Ether networks frequently interface with the internet, and a lot of preparation and protection has to happen to make sure the data can go halfway around the world and arrive in good shape.

Standard Ethernet works at 10Mbit/second, 100Mbit/second and 1Gbit/second data rates. Newer, faster equipment is usually backwards-compatible, meaning it can be used with older, slower equipment. 1Gb Ethernet often uses **jumbo frames** with up to 9kB of data.

In self-contained computer systems, RS-232 has largely been replaced by USB. In industrial and large office applications, RS-232 is being replaced by Ethernet instead, partly because Ethernet works over longer distances than USB. The simplest Ethernet systems work to a distance of 100 feet, but when fiber optics are used, the maximum working distance is 1,000 feet or greater.

The most common physical design uses unshielded cable with 4 twisted pairs, terminated by 8P8C (RJ45) connectors. Common versions are **10BASE-T**, **100BASE-TX** and **1000BASE-T**. The leading numbers indicate the bit rate: 10MB/S, 100MB/S and 1GB/s respectively. "Base" means the data is sent as a baseband signal (as in baseband video, not modulated on a carrier), "T" means twisted, unshielded pairs, and "X" identifies a particular version. Data is sent as a balanced signal. The cable used is an improved version of standard 4-pair telephone cable; the only thing that's special about it is that it has low capacitance and crosstalk. This is known as **category 5** or **CAT5** cable.

An improved version called **CAT5e** is now standard and regular CAT5 is essentially obsolete. **CAT6** cable is even better, used for runs that approach the official maximum length of 100 meters.

For 10BASE-T and 100BASE-TX, only two pairs are used. A device transmits data on the pair connected to pins 1 & 2 and receives on the pair connected to pins 3 & 6. There is an "A" color code and a "B" color code; the difference is that the green and orange pairs are swapped. This means one end's "transmit" pins are connected to the other end's "receive" pins, which is how you'd expect it to work. This is called a **crossover cable**.

Chapter 6 page 26

A crossover cable (A to B) is used to connect two devices together; a straight-through (A to A) cable is used as an extender. You have to be careful to make sure you're using the right one.

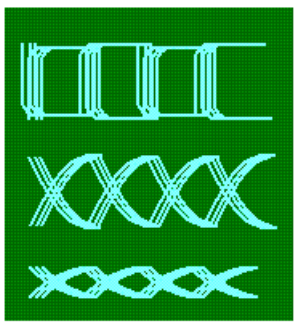
Sometimes, the two spare pairs are used for other things – a second 10Mb or 100Mb channel, DC power (**POE**, **power over Ethernet**) or analog telephone lines. Caution must be exercised because most ethernet equipment terminates these unused lines. These are non-standard uses so you have to be sure what you're doing.

10BASE-TX binary voltages are +2.5V and -2.5V. As with USB, these are differential voltages, measured across the pair. 100BASE-T uses three voltages: +1V, -1V and 0V. You'd think these small voltages (compared to RS-232) would make the system more vulnerable to interference, but the use of balanced twisted pairs makes them practically interference-immune.

With 1000BASE-T, also called **gigabit Ethernet**, **GbE** or **1 GigE**, all four pairs are used, each pair is **bidirectional** and there are five voltage levels: +1V, +0.5V, 0V, -0.5V, -1V. These are the actual voltages on the line; as with video, the sending units put out higher voltages that are dropped by build-out resistors to match the line impedance. 1GigE capability is now pretty standard in laptop and desktop computers.

With higher data rates and longer cables, the voltage transitions "smear" due to cable capacitance and inductance. On an oscilloscope, if 1s and 0s are overlaid (as they will be when the scope repeatedly scans a continuing data stream), the display resembles eyes which are increasingly closed as smearing gets worse.

This is shown in the illustration to the right, which may be confusing at first glance. "A" represents multiple scans of a data stream, where flips and flops are overlaid. If you separated the waveforms, you'd see a number of square waves. When overlaid, they appear to have rectangular "openings." "B" represents the same data with high frequencies attenuated by a cable. "C" shows severe high frequency attenuation; the openings look like eyes that are almost closed.



When the square waves are rounded off this severely, it becomes difficult to tell where the flips and flops are. Equalization (discussed in EM3) is routinely used to counteract smearing and restore distinct voltage transitions, extending the maximum practical cable length.

Nowadays, higher data-rate equipment automatically tests the cable before the first data communication. The transmitting end sends a standard bit pattern and the receiving end compares what it receives to what it knows the transmitter sent. Then it makes automatic adjustments to correct for level and high frequency attenuation in the cable. In a sense, it **learns** the cable's characteristics and compensates for them.

When building an ethernet system, you can't always rely on this automatic learning. Sometimes the system components aren't built with the learning function, or cable losses will exceed the system's ability to compensate. You often don't know you have a problem until after the cable has been run through walls etc. In such a case, “aftermarket” equalization can often avoid an expensive rewiring job. Somewhere between send and receive ends, you cut the cable, put connectors on, and insert an in-line amplifier/equalizer unit. Like any amplifier, this unit has to be powered. If you're lucky, you'll be dealing with a cable with a spare pair or two, and you'll be able to use POE. Otherwise you'll have to run a separate DC line or find and outlet to plug in a wall wart.

Are we done? Nope. The hot thing now is 10Gb Ethernet, also known as **10GE**, **10GbE** or **10 GigE**. There are already tens of millions of 10 GigE ports in use. It was first implemented using fiber optic cable or **twinax**, which is RF cable containing a balanced pair plus shield. But today, simple unshielded twisted pairs can do 10Gb to 100 feet. 40Gb and 100Gb are coming.

The key to making this work is to encode the data in such a way that the actual signal frequencies are much lower than the number of bits per second that they carry. This would seem to violate the Nyquist sampling rule we talked about in EM3, the one that says the maximum frequency that can be digitally encoded is half the digital sampling or clocking frequency. One of the ways this is done is to abandon binary levels in favor of multilevel digital logic. With two levels, two bits can encode four values. With five levels as in 1GigE, two “bits” can encode 25 values. That's equivalent to between four and five binary bits.

So apparently I was wrong, back in chapter 1, when I talked about how efficient computing is with two levels? Not really; we're not talking here about *computing*, where binary is ideal, but about *communicating*, where it isn't. There is a bit rate (bits per second) limitation to any communication channel. If each bit can carry more information, the information rate can be increased. There are a number of ways of doing this, which we'll talk about in EM6.

Not all Ethernet uses unshielded twisted pairs. When a net is extended to several buildings, e.g. the entire campus of a university, it becomes a **wide area net** or **WAN**.

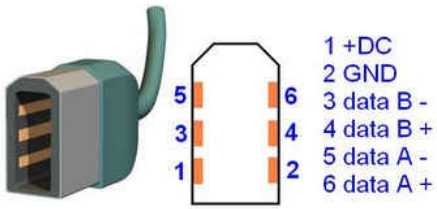
The distances involved are beyond the capabilities of twisted pair wiring. In these cases, coax cable or fiber optics are employed. This may change. Engineers are getting plain old unshielded twisted pairs to do things far beyond what anyone thought possible.

FIREWIRE / IEEE 1394

Firewire is a high-speed interface developed by Apple in the 1990s and formalized as industry standard IEE 1394. Its defining characteristics:

- serial data transmission
- daisy chain configuration, like MIDI except that communication is bidirectional
- standard data rates of just under 100, 200 or 400 Mbit/s, identified as S100, S200, S400.
- maximum length of 4.5 meters, though the repeater amplifiers in daisy-chain systems permit total length of 72 meters
- up to 63 connected devices
- a 6-pin connector, which has come to be known as the “alpha” connector. Two pins supply power to small peripheral devices. The standard provides for 30V at up to 1.5A, but this varies a lot – laptops often supply only 9V. You could compare this to Ethernet with two wires added, so DC power is automatically provided for.

Apple computers have Firewire connections, as do many others including my old Windows machine. Sony has a version called “iLink” which uses a 4-pin connector that does not supply DC. Texas Instruments' version is called “Lynx.”



IEEE 1394 connector
(cable end)

IEEE1394 continues to be developed. There is now a 9-pin version, called the “beta” connector, that will do just under 800Mbit/s. IEEE1394b lays out specifications for 3200Mbit/s Firewire, while IEEE1394c implements 800Mbit Firewire using standard, inexpensive 8P8C (RJ45) connectors and CAT5e cable. Data rates up to 6.4Gbit/s (800MBytes) are expected in the future.

Adoption of Firewire has been hampered to an extent because it's proprietary and manufacturers must buy licenses to use it, and in any case things like a keyboard or mouse have no need of its high data rate. However, the next generation, called **Thunderbolt**, has the backing of Intel, which means a lot, and it's open source. Thunderbolt, also known as **LightPeak**, is limited to 3 meters and 7 daisy-chained devices, but it will do 20Gbits/s. It's standard on newer Apple and Sony machines.

Thunderbolt was originally intended as a fiber-optic system, until it was shown that a 20Gb/s data rate could be achieved over unshielded twisted pairs. The connector includes DC power pins, permitting manufacturers to build amplifiers right into the connectors. This in turn permits an increase in distance to about 30 meters. In the not too distant future, fiber optic cables will be available with electrical-to-optical conversion circuitry right in the cable.

All of this means Thunderbolt cables of all lengths, wired and optical, will be interchangeable and pluggable into the same connectors.

A 20Gb/s standard was unimaginable a decade ago. Does anybody really need it? Maybe so. Now that we have developed technology to process and store huge data files, the bottleneck is the time it takes to move files from one place to another. For example, it's a good idea to occasionally back up the information on a hard drive by copying its entire contents to an external drive. With a terabyte drive, this takes all day via USB. Thunderbolt will do it in a few minutes. The other reason we need 20Gb/s is video. 4k horizontal pixel displays are widely available now, and they're testing 8k in Japan. Thunderbolt has the necessary bandwidth to do at least 6,500 horizontal pixels.

Thunderbolt uses the same 20-pin connector as the **displayport** video standard. There are four parallel, balanced data channels called **lanes**. I'm not going to show the pinout because there are some complexities and end-for-end swapping involved; you can find detailed information on the web.

LVDS

Many of the more recent techniques are similar in that they use balanced signals on pairs of twisted, unshielded wire. The push to unshielded pairs has been driven by a desire to lower costs, but perhaps more by a desire to avoid the higher capacitance of shielded pairs, which makes high data rates harder to achieve. Also, as we noted in EM3, shielding starts losing its effectiveness at some distance from a ground point – and at high data frequencies, "some distance" means inches.

LVDS means **low voltage differential signaling**. It's a communication standard in and of itself, and the concepts on which it is based are part of DVI, HDMI, Thunderbolt and other differentially based systems.

Despite the "low-voltage" in its name, LVDS is best thought of as a current-based system, like MIDI. A constant current of 3.5mA flows from the sending unit to the receiving unit, through a cable-terminating resistor (typically 100Ω for CAT5 cable) and back to the sending unit.

The current thus forms a loop, a classic closed circuit. There is always current flowing, but the direction switches to indicate a
1 or 0.

At the receive end, the 3.5mA through 100Ω develops a 350mV voltage drop which is read by a differential input; thus the “differential” in the name, which is equivalent to “balanced” in analog signaling. 1-3Gb/s data rates are common.

There are a number of variations on LVDS, and other standards that are similar. For personal and small-office networks, systems seem to be converging on unshielded, twisted, balanced/differential transmission at data rates once thought impossible for twisted pairs. LVDS has been the standard for communicating between the “guts” of a laptop and the display. Some systems are migrating to Thunderbolt for internal communications, but new LVDS products continue to be made.

CAN BUS

The **CAN** or **controller area network** bus was developed for automobiles, but is now showing up in machine control and other applications. Maximum data rate is 1Mbit/s up to 40 meters; lower data rates permit longer distances.

The most striking feature of CAN is the lack of a central computer. Individual MCUs all over the vehicle attach to a common bus. There is no central authority to arbitrate bus clashes. Instead, various nodes have differing priorities. If two nodes transmit at once, the node with the higher priority overrides the lower-priority node.

The latter senses this, stops transmitting, and waits until the higher-priority message is finished. The CAN system is a system of rank, like the Army, where each rank's authority overrides the rank below.

A CAN bus has four wires. Data is differential and sent over a twisted pair of wires, along with 0V and +DC wires. In the electrically noisy environment of a car, it's often best to put an overall shield over the wires. There's enough variation among different systems that it's difficult to give you more specific information – so I won't.

At this point, having reached the end of the discussion about common transmission standards, I am not satisfied, I feel that I haven't adequately covered the subject. There are so many methods and so many variations that it would be impossible to cover them all, and new ones are being developed all the time. But no matter how strange or exotic a system, it still has to obey the basic electrical rules you've learned. A new system must be a variation or combination of the techniques already used in existing systems.

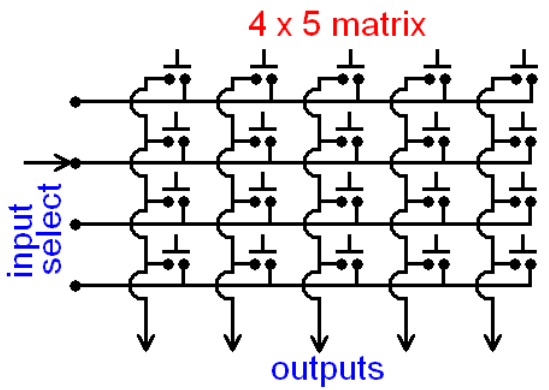
We've looked at computers and at their interconnections, and at a few of the peripherals. Before I close this chapter, I want to look at the three items – peripherals in a desktop system, built into a laptop or tablet – the **user interfaces** that turn a computer into something that's actually useful.

THE KEYBOARD

The keyboard has an MCU of its own which scans the keys, converts any letter or number that may be pressed to an ASCII character, and sends it to the computer via, usually, the USB port. The most common key arrangement is called a **QWERTY** keyboard because those are the first five letters at the upper left corner.

Each key is a switch that connects two bus wires. A typical keyboard is arranged in six horizontal rows with about twenty keys each in most of the rows. The simplest wiring system would be a 6x20 **matrix,** six horizontal wires and twenty vertical ones, with a pair of contacts at each crosspoint. (The illustration is a simpler 4x5 matrix.) The MCU would have an output port of six wires and an input port of twenty wires. Hundreds of times per second, it would put a “1” (or “0” if the designer prefers) on each of the six horizontal wires, one after the other.

Any time a “1” showed up on one of the vertical wires, the MCU would know that a key was being pushed. The combination of specific “1” output and resulting “1” input points the key.



We've seen some of this before. The matrix is similar to pinpointing a data error by arranging bytes into blocks and using vertical and horizontal parity bytes. In a keyboard, when the MCU makes one of the horizontal wires go high and sees a resulting “1” on one of the vertical wires, it knows which key is pressed. It converts this to an ASCII byte and sends it on to the computer. We talked about multiplexing in an earlier chapter; this is an example.

As long as only one key is pressed, this system works fine. But human typists often accidentally press two keys, or press a second before letting off the first. This can lead to the phenomenon of **phantom** keys, where an electrical “sneak path” seems to indicate that a key is being pressed when it isn't.

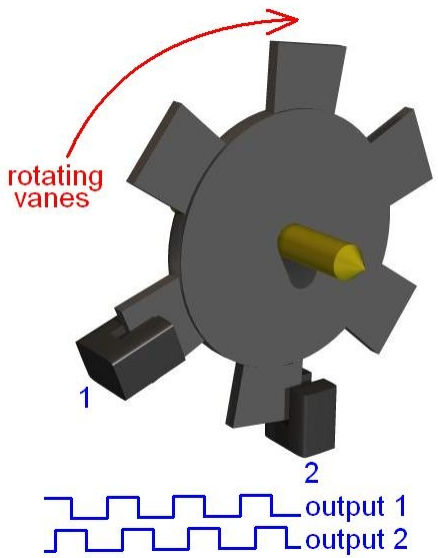
For example, if the key at X=1, Y=3 is pressed at the same time as the key at X=3, Y=1, the computer will see both keys, and sneak paths around the shorted circuit will seem to indicate that X=1, Y=1 and X=3, Y=3 are also pressed.

The easiest way to deal with this is to simply freeze the system, preventing it from accepting any input when more than one key is pressed. A better way, well within the capabilities of the keyboard's MCU, is to analyze the timing, noting the order in which closures occurred, and the closure pattern, using knowledge of which dual-key situations can result in which phantom key closures.

Another is to reconfigure the buses so the most common problems are avoided. If you ever take apart a keyboard and trace the buses, you'll find that they aren't straight vertical and horizontal lines. They snake all over the place in a way that seems to make no sense. They actually do make sense; they're arranged to minimize the most common phantom key problems. Here again we run into that great advantage of digital. Each copy of an a complicated analog circuit costs more than each copy of a simple one. On the other hand, each copy of a complicated digital program costs the same as each copy of a simple one. It's more expensive to develop, but once developed, duplicating it in a million keyboards is free.

Combining all available techniques, a keyboard can very effectively compensate for imperfect human typists and accept input without getting hung up and slowing the process down. Of course, the program must also be smart enough to deal with situations where it is proper for two keys to be pressed, as in Shift + any key.

Digression: many people think bus should be spelled “buss,” probably because they've seen “Buss” fuses made by the Cooper-Bussman company. It's “bus,” plural “buses.” This message brought to you by Nitpick and Quarrel, makers of fine distinctions since 1750.



THE MOUSE

A traditional mouse senses position by rolling a ball around on a surface. Two shafts touch the ball, and as the ball spins, the shafts rotate in longitude and latitude. A disc with vanes is linked to each shaft, and a pair of optical interrupters at each disc counts the number of vanes that pass by.

Why a pair of interrupters? You have to know which way the disc is turning. They are spaced so that, whenever one interrupter is transitioning from 1 to 0 or 0 to 1, the other will be fully on or off.

This results in a pair of square wave trains that are roughly 90^{O} apart, as shown in the illustration, which is called **quadrature**. The MCU can use these to tell which way the disc is rotating.

Optical mice do about the same thing in a different way. Several optical detectors “see” small bumps and irregularities in the surface that the mouse moves over. The MCU compares data from the sensors to tell the time between when one sensor sees a bump and another sensor sees the same bump. From this it can compute how fast and in which direction the mouse is moving. A *very* complicated program. But, as previously mentioned, a complicated program can replace highly complicated physical circuitry, and be duplicated for free, bla bla bla. That, perhaps more than any other factor, is how digital technology has been able to make highly sophisticated stuff cheap, like the $5 keyboard I'm typing on.

THE MONITOR

There are few computers that don't have a monitor, either built-in or as a peripheral. After all, it's hard to drive if you can't see the road. But I'm going to save that discussion for its own chapter, because there's a lot to say about monitors. Also, printers are essentially monitors whose images are permanent; they use a lot of the same technology. So I've reserved all of chapter 9 for display devices.

COMPUTER APPLICATIONS

The most common computer applications are

- to control things
- to compute things
- to display things

These functions tend to involve something I mentioned in chapter 1, representing analog quantities as digital numbers. So, here's where we're going next: chapter 7 is a description of how computers convert analog signals to digital and back to analog, and the methods and benefits of the mixed analog-digital approach. In chapter 8, we look at computers that control machines, and then, in chapter 9, I'll bring the discussion back around to monitors and other kinds of displays, including printers.

I began chapter 5 by saying our study of MCUs and computers would just scratch the surface, and that's what we've done. It is much more important for most techs to understand how a system is put together and how the parts relate to and communicate with each other, than it is to have extensive knowledge of what's going on inside each part. Just as an opamp is intended to be treated as a black box, a computer or a program can be regarded as a package that you just turn on and use.

Employers are urging educators to de-emphasize teaching electronics at the component level and concentrate instead on system-level electronics. I'll go along with that for the most part, but I have to say that the sharpest system-level techs I know, the ones other techs admire, are the ones who make a habit of digging deeper. Studying what others have done both in software and hardware is worthwhile. You run across clever ideas, and from then on they're part of *your* bag of tricks, *your* ability to solve problems that stump others – tricks you would never have thought of on your own. Becoming a wizard-level tech is mostly a matter of having a curiosity that makes you learn stuff just because you like to know.

End of Chapter 6

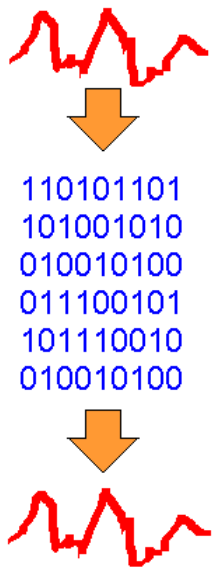
EM4 Chapter 7
MIXED DIGITAL / ANALOG

Objectives of this chapter:

- *define “digitization”*
- *discuss advantages and disadvantages of digital representation of analog signals*
- *discuss quantization and quantization error*
- *define “monotonic”*
- *define “sample,” “sampling rate,” “bit depth”*
- *discuss the Nyquist theorem and Nyquist frequency, aliasing, and practical maximum signal frequency to sampling rate ratios*
- *explain nonlinear quantization and the reason for it*
- *discuss DSP and transversal filters and give several examples*
- *discuss packets, latency in filters and latency due to buffering*
- *explain lossy and lossless data compression*
- *explain what “MPEG” stands for and give a brief summary of video compression techniques*
- *describe waveform generation via lookup table*
- *define “pseudorandom” noise and describe how it can be digitally generated*

There's a well-known principle in computer engineering called Moore's Law. First advanced by Intel co-founder Gordon Moore in 1965, it's not really a law but an observation: the number of transistors we can put on a chip of a certain size doubles every two years. This observation has held true for almost half a century, and will probably continue in the future (though perhaps at a slower pace, doubling every three years).

Yet even if digital technology kept getting more powerful forever, we still wouldn't be able to do all the useful things we can do with it if we hadn't mastered the science of *converting* analog voltages to digital and back again.



After all, we live in an analog reality. Sight, sound, taste, smell and feeling are all analog. The text you're reading is stored in the computer as ascii Bytes, but it has to be converted to letter shapes for you to read it. To the computer, the difference between “big” and “dig” is the difference between 01100010 and 01100100. To you, it's which direction the rounded part of the first letter goes.

The ability to **digitize** analog signals set off what is commonly called the “digital revolution.” Digital representation of analog signals, specifically audio, has long been used in the telephone system, but what really got things going was the introduction of the audio CD in 1982.

The music business really needed digital. Way back in the late 1940s, it was possible to make analog tape recordings that are still high fidelity by today's standards, and wax discs that are nearly as good as far back as 1929. But in a 1980's recording studio, microphones were fed to as many as 32 separate tape tracks, then mixed down to two tracks on another tape recorder, and perhaps two or three generations of tape before being cut into a vinyl disc, with a small loss of sound quality at each step.

And then there was analog's great enemy, Time. Analog tape deteriorates. Today, thousands of classic rock tunes from the last half of the 20th century can only be heard in the processed form that was released to the public. It is often impossible to go back to the master tapes and do a newer, cleaner mix because the tapes have disintegrated.

The recording industry jumped on digital technology as soon as it was available, not for the sake of better sound, but for the sake of durability. (And for the sake of having something new to sell.) The process of converting an analog signal to digital involves its own kinds of deteriorations, but once the signal has been converted to bits, further deterioration stops. In theory, the signal can be processed as much as desired or stored for decades with no loss of quality.

Unfortunately, the technical standards for the audio CD were set in concrete before digitization was fully understood. Continuous sine waves were used as models, and if there's anything you can confidently say about music, it is that music is *not* continuous sine waves. Problems were built into the CD standard that had not been fully solved even by the time CDs started to become obsolete.

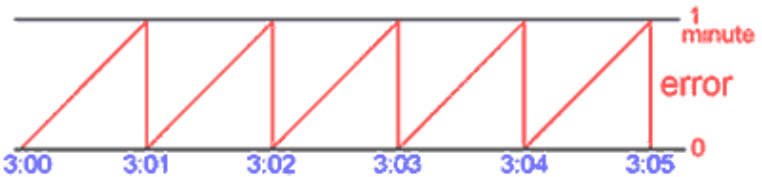
But those problems are behind us. Today, almost any analog signal can be stored and processed digitally, and the level of quality is a matter of choice, not technical limitations.

In this chapter, I'm not going to get into the transistor-by-transistor technology. A tech seldom deals with digital signals at that level. Instead, I'm going to discuss what digitally represented analog is all about and how the various parts of it work.

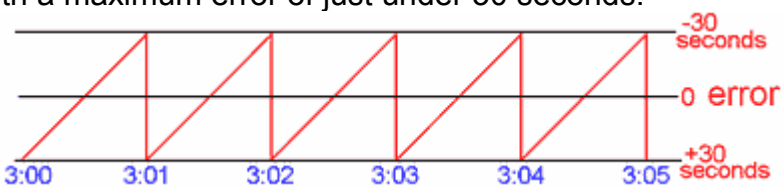
SAMPLING IN THE VOLTAGE DOMAIN

If you look at a digital clock, what you're seeing is the continuous flow of time, chopped up into chunks. If the clock only shows hours and minutes, each chunk is one minute wide. And, most of the time, the number displayed is wrong.

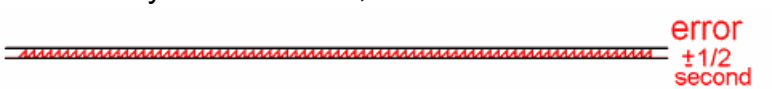
Suppose our clock is perfectly synchronized to the master timekeeping reference, which in the USA is an atomic clock operated by the federal government in Boulder, Colorado. At the precise moment when its display changes from 3:00PM to 3:01PM, it is telling the full, exact truth. An instant later, and for the remainder of that minute, it's lying. At various moments, the actual time is 3:01:20, or 3:01:35, or whatever, but the clock keeps saying 3:01. It won't be correct again until the moment when it changes from 3:01 to 3:02.



The maximum error occurs when the time is almost 3:02; the clock is still showing 3:01, so it's almost a minute off. This error is always on the late side. If we were to reset the clock so its transitions came 30 seconds early, then the indication would be early half the time and late half the time, with a maximum error of just under 30 seconds.



If the clock showed seconds, then instead of time being chopped into minute-long chunks, time would be chopped into second-long chunks. The clock's maximum error would be reduced by a factor of 60, to half a second.

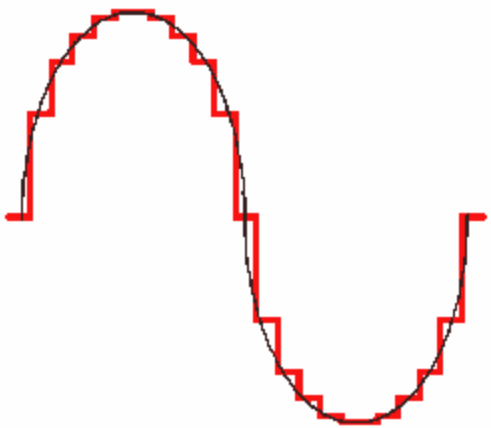


I've used the digital clock to highlight an unavoidable truth: a digital representation of time, time spelled out in numbers, is wrong most of the time. The more digits we add, the less wrong it is, but it's almost never right. An analog clock, one with hands, doesn't have this problem. At every instant, even when the hands are between numbers, it indicates the correct time (assuming it's set correctly). But numbers, simply by virtue of the fact that they are discrete quantities, cannot precisely represent time, which flows continuously. When time is represented by numbers – when it is **quantized** – there is unavoidable **quantization error**.

All digital representations of analog values suffer from quantization error. In effect, we're always truncating. You remember truncation from the math chapter of EM1: when a quantity cannot be precisely written in numbers, no matter how many places you calculate it to – e.g. "pi" or the square root of two – or, when we simply don't need absolute precision, we "truncate" by writing out only as many digits as we need to get the precision we want.

In the case of a real-world analog quantity like voltage, we have no choice but to truncate. There's no such thing as 1.0000000000... volt. And even if there were, there's no such thing as a measuring instrument with 100% precision. So we do the best we can, or the best that it's worth doing in a given situation. A tech should always keep in mind that *any* number that represents an analog quantity is imprecise to some degree.

When we're dealing with a digital representation of an analog voltage, we label the imprecision "quantization error." In the illustration at right, quantizing error is the difference between the analog signal (black) and the digitized signal (red). This would be considered rather crude quantization, suitable for walkie-talkies etc.



In EM3, we noted that an imperfect analog signal can be viewed as the algebraic sum of a perfect signal and a certain amount of grunge. When an analog value is represented by a binary number, maximum error is ½ the significance of the least significant bit.

Does this unavoidable error mean digital is automatically inferior to analog? Not at all. Analog circuitry has its own inaccuracies, and unlike digital, each additional circuit the signal passes through adds more error. When digital quantization is carried out to 20 binary places, maximum error is about 1 part in a million. Few analog circuits are that accurate.

BIT DEPTH AND SAMPLING RATE

When we go to represent analog voltages digitally, the first question we need to answer is, how many bits? How accurately do the analog values need to be quantized? When the voltage we're measuring is AC, there's a second question: how many measurements do we have to take per second to get an accurate representation of the waveform?

Suppose we're trying to quantize a telephone conversation, where the waveform changes up to 3,000 times per second. Taking one measurement every half second isn't going to cut it.

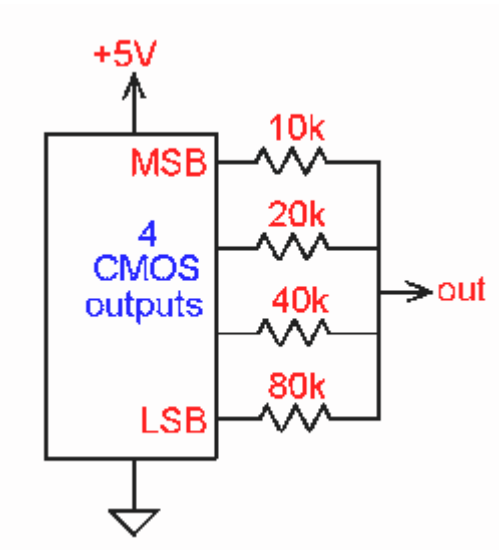
The process of taking digital “snapshots” of changing analog values is called **sampling**. To restate our two questions: how often do we have to sample, and how accurate does each sample have to be? How finely do we have to slice the waveform, both in time (X-axis) and voltage (Y-axis) dimensions, to get pieces small enough to digitally represent the waveform with the accuracy we want? There's no single answer. A movie sound track requires more and better samples than a telephone conversation.

The number of samples per second is called the **sampling rate**. The precision of each sample is called the **bit depth**.

Let's approach the problem backwards, turning a digital word back into an analog voltage.

1101_2 is a number, but when it lives in CMOS, it's also voltages: 5V, 5V, 0V, 5V. If we sum the outputs through resistors as shown below, and if we ignore minor complications like the fact that the CMOS outputs have a bit of resistance of their own, we can get 16 different voltages in equal increments that correspond to the 16 states available from four bits:

0000	**0V**
0001	**0.33V**
0010	**0.67V**
0011	**1V**
0100	**1.33V**
0101	**1.67v**
0110	**2V**
0111	**2.33V**
1000	**2.67V**
1001	**3V**
010	**3.33V**
1011	**3.67V**
1100	**4V**
1101	**4.33V**
1110	**4.67V**
1111	**5V**

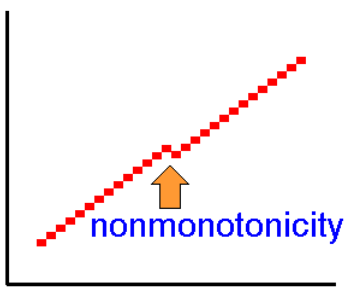


Note that the resistor ratios are in multiples of 2, corresponding to the fact that each digit has twice the “weight” of the digit next to it.

If we were to add a high-current, unity gain amplifier, we'd have a digitally programmable linear power supply.

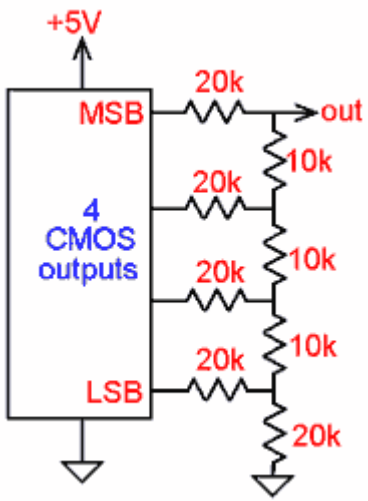
D-A CONVERTERS

The circuit on the previous page is a simple **D-A** or **digital to analog** converter. For some applications, it's all you need. The resistors that represent more significant digits must be highly accurate. The MSB, bit 4, is low up to a count of 0111 and then switches high at the next count, 1000. If that resistor is off by 5%, the voltages around that transition may be off by as much as 0.125V, which is significant. If we use more bits, so as to obtain a more precise digital representation of the analog voltage, we may find that the error in the MSB resistor overwhelms the least significant bits. The rising output voltage characteristic may even go backwards between 01111111 and 10000000. This causes all sorts of problems, especially in feedback and closed-loop servo situations. A decent D-A may have up to ½ LSB error, but if the output doesn't go backwards, if a rising count never results in a falling voltage, that's a **monotonic** characteristic. If it does, that's **nonmonotonicity**, which is a great word to impress other people with. It's also a good sobriety test. If you can't say it, you shouldn't drive.



There's also the practical problem that standard resistor values usually aren't 2X multiples of each other. But then, few people actually make D-A's with discrete parts. When a D-A is integrated, the resistors tend to have accurate ratios simply because they're made at the same time on the same substrate within thousandths of an inch of each other.

An easier arrangement from a manufacturing standpoint is to use a 1R / 2R ladder network. In this design, each build-out resistor is the same value as the next, but the current contributed by resistors farther from the output is attenuated. With this design, it's not too difficult to make an 8-bit D-A where the worst error is less than half the size of the least significant bit. That goal – ½ LSB or better linearity – is a kind of benchmark for a "proper" D-A.



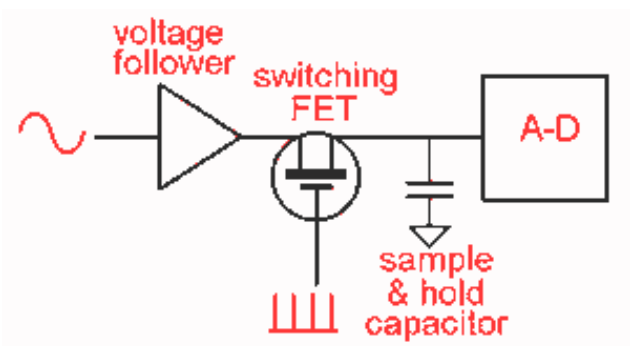
Where it is important to avoid momentary voltage glitches as various outputs flip and flop, a gated latch circuit may be added to insure that all bits change at once.

Now let's go the other way: let's sample an analog voltage and convert it to a digital value.

A-D CONVERTERS

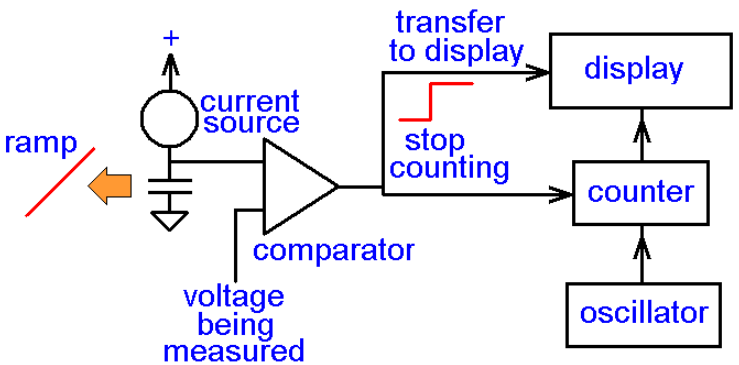
Converting a digital signal to analog is fairly simple. Going the other way, we get into some complication.

With most **analog-to-digital** converters, the calculation takes time, and we can't have the analog voltage changing while we're calculating it. So the first thing an A-D converter does is take a "snapshot" of the analog voltage. One way to do this is to briefly connect the voltage across a sample and hold capacitor, so the capacitor acquires a charge equal to the voltage at that moment. Then, the A-D converter reads the voltage on the capacitor and converts it to a digital value. The voltage follower isolates the signal source and keeps the capacitor from loading the source during sampling.



Counting (Integrating) A-D

The most basic A-D is based on an oscillator and a ramp generator, and is typically used to measure DC voltages. During measurement, a current source charges a capacitor, generating a ramp. At the same time, a ripple counter counts the oscillations of an oscillator. When the ramp voltage equals the voltage being measured, the comparator flips, counting stops and the number at that moment is transferred to a display.



This is the setup typically used in digital voltmeters. It has the advantages of simplicity, low cost, and the fact that every step is automatically equal to every other if the ramp is linear. There is no glitch between 01111111 and 10000000, and the converter is automatically monotonic. Its disadvantage is the long time it takes to quantize a sample.

Successive Approximation

The **successive approximation register** or **SAR** achieves the same result in a lot less time, at the cost of some added circuit complexity.

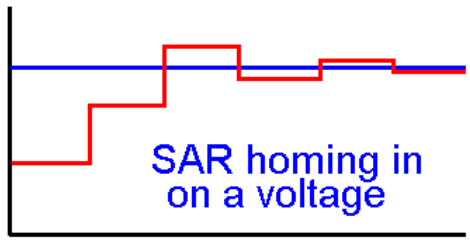
You've probably played “20 Questions,” the game where you try to guess an object by asking no more than 20 questions. You start with no idea what the answer is, so your first question is something general like, “organic or inorganic?” Eventually you get to something very specific like “sea squirt.” (A sea squirt is an ocean creature that seeks out a rock to attach to. Once attached, it never needs to think again, so it eats its brain. This also happens to some professors after they achieve tenure.)

The SAR works like 20 questions. It makes estimates, each twice as precise as the last, homing in on the correct value. For example, let's say the supply voltage is 8V, and the voltage being sampled is 6.7V. The SAR will first ask the question: Is the voltage above or below 4V (half the supply voltage)? It's above, so the first binary bit will be 1.

Second question: Is the voltage above or below 6V (halfway between 4V and 8V)? Above; second bit: 1.

Third question: Is the voltage above or below 7V (halfway between 6V and 8V)? Below; third bit: 0.

Fourth question: Is the voltage above or below 6.5V (halfway between 6V and 7V)? Above; fourth bit: 1.

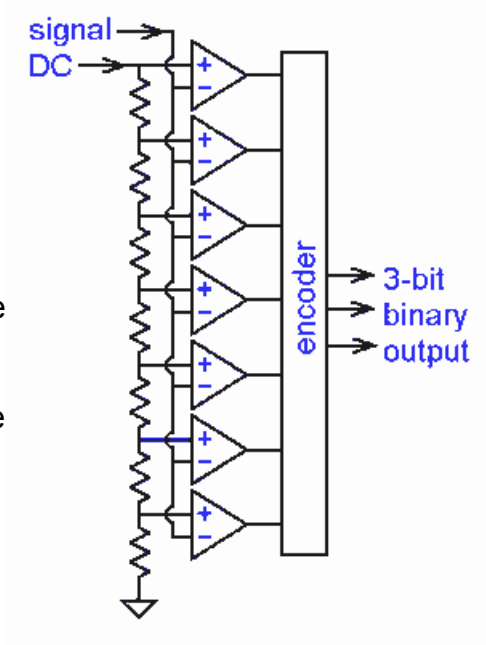


With each step, the SAR gets closer to the correct voltage, building a binary word one bit at a time, which currently stands at 1101.

One common way to build a SAR is to use a D-A to create a feedback signal to the A-D. The D-A makes a guess, the A-D creates a resulting output voltage, and the D-A uses the error between that voltage and the voltage being sampled to make the next guess.

The SAR suffers from the same problem as our primitive resistor-based D-A: any error in the most significant resistors will upset the linearity of the quantizing characteristic. For this reason, it isn't a good choice where the signal spends a lot of time in the neighborhood of the 01111111 – 10000000 transition, which when we're dealing with an AC waveform is the area near zero volts. Audio in particular doesn't work well with SAR D-A's, where the transition glitch sounds a lot like crossover distortion in class B amplifiers. It's especially annoying because as the sound gets quieter, the distortion gets worse, like a bad MP3.

Flash A-D



A Flash A-D consists of a large number of comparators and a voltage divider with many taps; each comparator goes high or low depending on whether the input signal is above or below its specific divider tap. The outputs of all the comparators are connected to a circuit which finds the highest-value comparator in a "1" state and outputs the corresponding digital word. There is no clock and no timing. The binary output changes almost instantly with changing input signal; thus "flash," not to be confused with flash memory, which is something entirely different. It is always monotonic.

The Flash A-D's big problem is complexity; a separate comparator is needed for every possible value. The illustration shows a flash A-D with a bit depth of 3 bits, or one of eight possible values. Digitizing to 8-bit precision requires 255 comparators, and that number is almost doubled with every added bit. This isn't an insurmountable problem with ICs. Fast flash registers can now digitize to 10 bits at a rate of 10^7 to 10^8 samples per second, which is necessary to digitize hi-resolution video.

Delta A-D

Where the input signal doesn't change much between samples, we can get rid of a lot of counting if, instead of measuring the voltage at each sample, we measure only the *change* in voltage from one sample to the next. In other words, we measure the difference or delta. 24-bit samples can be represented with 4-bit words, as long as changes are small enough that the difference between adjacent samples never exceeds 4 bits. We can insure that by sampling more often.

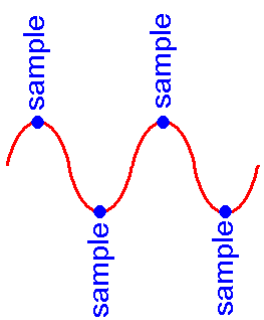
Taking that to its logical conclusion, we could represent a waveform sampled to 24-bit depth with 1-bit difference samples, as long as we take those samples very, very often. Each sample comes down to the question, "Higher or lower this time?" That's a simple circuit. It's also a circuit that doesn't have the transition problem between 011111111 and 10000000, which is one reason audio people love it.

For a 1-bit A-D to digitize as precisely as a 16-bit SAR, a great many more samples must be taken – for high-quality audio, millions of samples per second; for video, maybe a billion. That doesn't mean we need to store all those samples. A process called **decimation** converts the 1-bit sample stream to 8, 16 or 24-bit words.

Counting, SAR, flash and delta D-A's all have uses. Which one is chosen for a particular application depends on speed requirements, linearity, cost factors and so on.

SAMPLING IN THE TIME DOMAIN

The minimum sampling rate is a less complicated issue, because we don't have a choice. There's a law called the **Nyquist Theorem** that says you need at least two samples per cycle to reconstruct a sine wave; thus the maximum signal frequency is half the sampling frequency. For example, if you're sampling a signal whose highest frequency component is 20kHz, you need at least a 40kSPS rate – that's 40 thousand samples per second. When the sampled signal is converted back to analog by a D-A, assuming there's a low-pass filter at the output to block the hard-edged digital transitions, the original waveform will be reconstructed.



The highest frequency that can properly be sampled and turned back into an analog sinewave is called the **Nyquist frequency**, and it's equal to half the sampling frequency.

Any signal higher than the Nyquist frequency will come out at a different frequency than it went in. For example, if the sampling frequency is 40kHz and you try to sample a 25kHz sinewave, it will come out as a15kHz sinewave. This is called **aliasing**.

The Nyquist theorem assumes the signal is an unchanging waveform. With real-world complex signals, things work better if you take a lot more samples. For practical purposes, the number of samples per second should be 2.5 to 4 times the highest frequency to be sampled.

VIDEO SAMPLING

If you were to display a picture on an LCD or LED monitor, and if you looked very closely, you'd see something like this. Each of the squares is a **pixel**, short for **picture element**, and the number of pixels in each direction

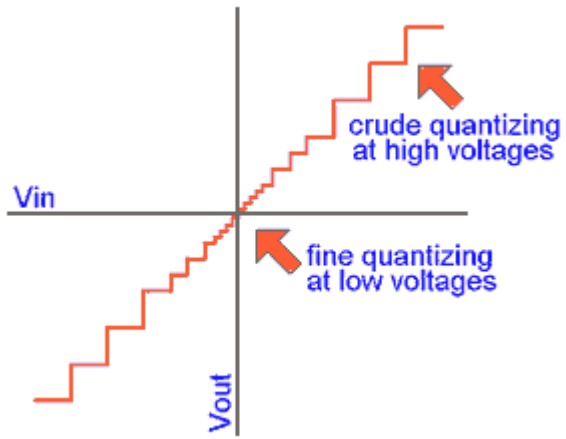
determines the horizontal and vertical **resolution**. It is obvious that the picture is being quantized into pixels in the vertical and horizontal directions, but each pixel also has its own quantizations. A pixel's brightness, called **luminance,** and its color, called **chrominance**, are determined by three voltages that represent the amounts of red, green and blue in the pixel, and in digital video these values are quanitized. If each color is quantized to a bit depth of 8 bits, that means a pixel can have any one of almost 17 million distinct combinations of color and brightness.

LINEAR AND NONLINEAR QUANTIZING

We could think of a digitized quantity as a little square, like a video pixel, with a width equal to the time duration of the sample and a height equal to the increment between one sampled level and the next – that is, height would depend on bit depth. Up to this point, we've been treating all these squares the same, as if every square were as significant as every other. In many types of instrumentation, that's the correct approach – every millivolt has the same significance. This is called **linear quantizing**.

But there are many other cases where a different digitizing scheme is called for. For example, since audio levels vary so much, it makes sense to quantize small voltages more finely than large ones. That way, instead of getting excellent quantization of large voltages and crummy quantization of small voltages, we get quantization that meets a certain minimum quality standard across a wide range of voltages.

The telephone system has used this kind of **nonlinear quantizing** for decades. Telephone companies are not aiming for excellent sound at any voltage level, but adequate sound at all levels. By varying the quantizing resolution depending on level, adequate quantizing can be achieved using a sample size of just 8 bits. Two similar quantizing schemes in use are **A-law** and **μ-law** encoding, which offer finer quantizing of small sounds at the cost of coarser quantizing of loud sounds – but about the same quantizing quality at all levels, *proportionately*.

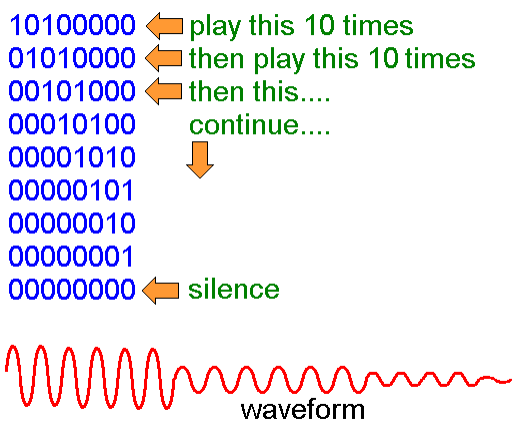


We now have everything we need for a basic analog/digital system. We already had computers that could input, store, retrieve and output binary data. Now we have ways of turning analog into binary and back again. But computers can do a lot more than just handle data; they can process it too. Let's see what happens if we apply some processing to the analog signal while it's in digital form.

DIGITAL SIGNAL PROCESSING (DSP)

Once analog signals are converted to numbers, we can perform any operation on them that can be described by a mathematical function. The simplest, probably, is to shift the data one bit right or one bit left, then feed it to a D-A converter and output it. Since every bit has twice or half the significance of the bit next to it, shifting the data left or right is equivalent to doubling or halving its voltage – in other words, boosting or attenuating by 6dB.

Suppose we have a bunch of samples, and the first sample is 10100000 as in the illustration at right. Suppose we start by clocking through all those samples 10 times, then shifting them right one bit, so the first sample becomes 01010000, and clocking through them ten times, then shifting another bit and so on. If the signal is attenuated 6dB with every shift, this is a way of getting a fade-out, like turning the volume control down. That scheme is often used to imitate the sound of a chime or doorbell.

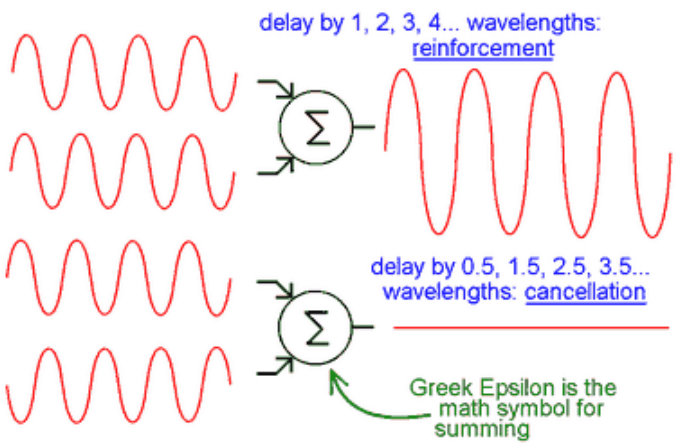


6dB chunks are pretty crude. If, instead of simply shifting the data, we multiply it by another binary word, we can change levels in increments much smaller than 1dB.

Another thing we can do is create a time delay. Analog time delays of more than a few microseconds are very hard to implement, but in the digital domain it's a piece of cake. You just write data to memory and then read it a little later. If you look at the TVs displayed at, say, WalMart, you'll note that the timing of the audio is different from one monitor to the next, leading to a massive echo effect. When a TV is set up in someone's home, one of the necessary adjustments is to set the audio delay so the audio is synchronized to the picture. Electronically, this is just a matter of changing RAM address indexes to adjust the time offset between write and read.

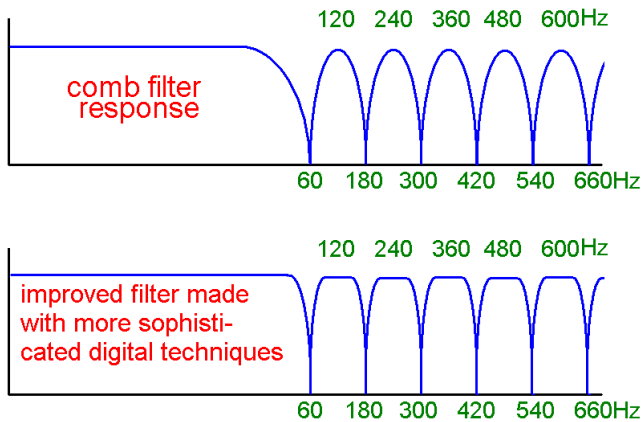
Where it really gets interesting is when we combine several reads at several different delays. In EM3, we discussed the fact that, if you delay a waveform by half a cycle, that's equivalent to flipping its polarity. In an analog circuit, this turns negative feedback into positive feedback and causes oscillation.

Since delays are easy in the digital domain, let's see what happens when we delay a signal by one millisecond and then combine it with a non-delayed signal. At a signal frequency of 1kHz, the delay will be exactly one cycle long, so the direct and delayed waves will add. They will also add at 2kHz, 3kHz, and every multiple of 1kHz.



But at 500Hz, the delay will be half a cycle. Positive alternations of the undelayed signal will coincide with negative alternations of the delayed signal, so they will cancel, leaving no signal. The same thing will happen at 1.5kHz, 2.5kHz, 3.5kHz and so on – that is, at 500Hz and every odd multiple of 500Hz.

The result is a frequency response with notches 1kHz apart. This is called a **comb filter**, both because the response looks like comb teeth and because it combs through the signal, removing a number of different frequencies.



Here's one possible application of a comb filter. Suppose we have an audio signal that's been contaminated by lamp dimmer buzz. If the dimmer waveform is symmetrical – if the dimmer turns on at precisely the same voltage during positive and negative alternations, which most dimmers

pretty much do – the interference will consist of energy at 60Hz and every odd multiple, or harmonic, of 60Hz. An analog filter circuit to notch out every one of those harmonics would be hugely complicated, but a digital comb filter can do it easily.

Another example: In NTSC color TV, the brightness information shows up mostly at 16kHz and harmonics of 16kHz. There's a lot of empty spectrum space between the harmonics, and that's where NTSC puts the color information. But if the picture includes a lot of almost-horizontal lines, like someone wearing a checkered shirt, brightness information starts to spill over into the frequencies intended for color. This causes the picture to go all screwy. The problem can be prevented by running the brightness information through a comb filter that won't let any signal through at the color frequencies.

We don't have to confine ourselves to a single delay, or to summing equal quantities. By summing differing amounts of signal with differing time delays, we can make a better comb filter with narrower notches, so we don't take such big chunks out of the spectrum.

We can also do all the usual things analog filters can do, and some things analog filters can't. For example, if you studied the filter chapter in EM3, you know there are different kinds of filters with response characteristics called “Butterworth” and “Chebyshev” and so on, and you know most of these filters “ring” on transients; the faster the rolloff, the worse the ringing. But a digital filter can have a very fast rolloff and little ringing, and that's useful. For example, due to a mistake in the original audio CD standard, early CDs sounded gritty. It was only the development of non-ringing digital filters that allowed them to sound good.

LATENCY

The cool things digital filters can do come at the price of an unavoidable time delay. Time-delay based filters are called **finite-impulse response** or **FIR** filters, also **transversal** filters. Because they're based on time delays, there must be a delay between input and output. This is called **latency**, and everyone who uses a cellphone is familiar with it. If you've ever listened to someone in the same room over a cellphone, so you can also hear their voice directly, you're aware of how much the cellphone sound is delayed. (In the case of cellphones, the latency is not due to filtering – we'll talk about that shortly.)

When transversal filters are used in the recording/playback process, latency doesn't matter. Even a football “instant” replay occurs 20 or more seconds after the original event. I have a digitized version of an analog recording of Enrico Caruso, who died in 1921. You could view that recording as

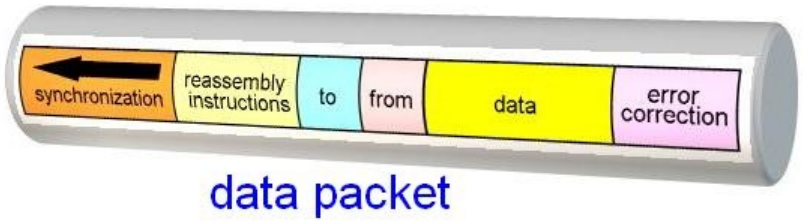
having a latency of 90 years. A few extra milliseconds isn't going to matter.

But in live performance situations latency becomes a real issue, not so much because of signal delay to the audience, but because of signal delay back to the performers. With multiple performers who have to be coordinated, e.g. musicians in a band, latency greater than about 5mS causes problems. Sometimes a separate, zero latency, all-analog monitor path is needed.

PACKETS AND LATENCY

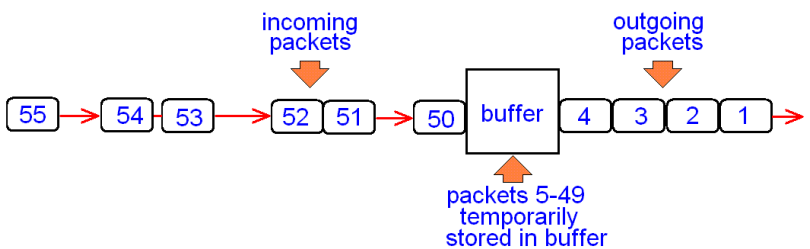
Once an analog signal has been converted to files full of digital words, there's no reason the files can't be split up and shuffled around, as long as each word (or batch of words) is attached to enough bytes of information to allow the system to piece things back together again. In fact, this is a very important part of DVD and CD technology. Data is scattered around, so a scratch in the surface causes a lot of little errors which can be corrected, instead of a big error which can't be.

A fragment of a data stream which includes reassembly instructions is called a **packet**. On the internet, and over the telephone, data is broken into packets so everybody's communications can be sent over the same channel. On the receiving end, a particular device collects the packets addressed to it and reassembles them into a continuous data stream.



data packet

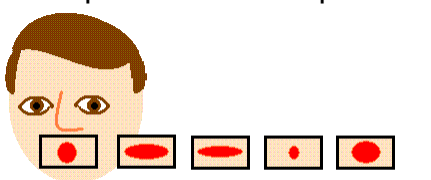
These packets do not always arrive at regular time intervals. To convert them into a properly timed output stream, it is necessary to collect a bunch before beginning the output. You recall the previous discussion of UART **buffers**, which are similar to what we're dealing with here. A buffer collects packets at its input and feeds them out at its output. A momentary disruption of packet reception won't cause a glitch in the output data stream if there's enough data in the buffer to tide the system over.



This means that there's a delay between data in and data out. The bigger the buffer, the greater the disruption the system can compensate for, but the longer the delay. That's what causes cellphone latency. There's also a delay in portable CD and DVD players, where movement is likely to cause disruptions in playback. If you've noticed that it takes forever to hear your music after you've pushed the button, that isn't because the system is slow. It's because there's a big buffer that's being loaded up with data.

DATA COMPRESSION

Most analog signals contain a huge amount of redundant, repetitive information. Think of a news anchor, sitting in front of a camera, reading from a Teleprompter. The only thing in that picture that's moving is the newscaster's mouth. But in traditional analog video, the whole scene is scanned over and over again, sending out a picture 30 times per second where once per second or once in ten seconds would suffice, except for that small part of the picture that is the newscaster's mouth. If we could eliminate the redundancy, think how much less bandwidth it would take to transmit the picture! More important, think how much less memory it would take to store, say, an hour's worth!



With analog, can't be done. But with digital, once everything's been converted to numbers and the data is socked away in memory, we can take our sweet time and process as much as we want, searching out and eliminating redundant information. That's what people mean when they talk about audio and video **data compression**.

Let's do a little math. A traditional DVD (not Blu-Ray) stores 700 x 576 pixel images, for a total of about 400,000 pixels per image. Each pixel requires three bytes for red, green and blue, for a total of about 1.2 MB per image. In traditional video, there are 30 images per second, which implies 36 MB per second. The most basic DVD stores just under 5GB of data. That works out to about 2½ minutes of storage without data compression. Compare that with the average actual DVD, which stores several hours of video, and you get an idea how much compression is being used.

For video, standard methods have been adopted by the **motion picture experts group**, which you know as **MPEG**. There are a lot of issues most people wouldn't have thought of. For example, it's nice to be able to go both forward and backward with video, which screws up transversal filtering unless you plan for it. Another example: old movies have lots of grain and visual noise, which change from one frame to the next. How does a processor tell that this is noise that should be ignored, not motion?

If you're getting the idea that I'm not going to delve into compression techniques in any detail, you're right. The technology used to compress video is extremely complex, and took a billion dollars to develop. But I'll give you a couple of hints.

Step 1: A video picture is made up of a whole bunch of horizontal lines. The illustration represents voltage-vs-time for one horizontal line. The voltage variations correspond to brightness variations at various points across the screen.

Instead of viewing this pattern for what it is – a moment by moment record of brightness – we could view it as a complicated waveform. In EM3 we noted that any waveform of any kind can be resolved into sine waves at different harmonically related frequencies and phase angles. This, in turn, can be reduced to an equation. If we convert the waveform to an equation, we have something that we can store more compactly.

Step 2: We apply a delta calculation vertically; that is, each line of the video picture is compared to the ones above and below it and we only record the difference information.

Step 3: We apply a delta calculation chronologically: each picture is compared to the ones before and after it. (In traditional video, there are 30 pictures per second, called **frames**, a term borrowed from film. In high-def, 60Hz is more common and some displays run at 120Hz.)

Step 4: We look for special opportunities to reduce data. For example, when a camera pans, the whole scene moves, but it's up to 100% redundant, just displaced right or left. If the system can figure that out, we can reduce the data again. This is where the really sophisticated computation takes place.

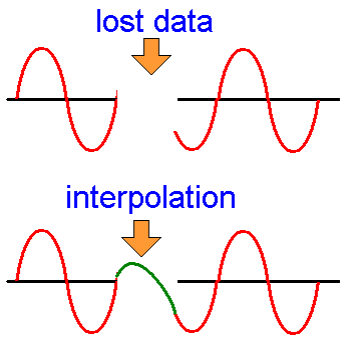
If you compare movies on DVD, you'll find that the amount of data taken up by different movies varies a lot, and the variation does not necessarily nave anything to do with the lengths of the movies. That's because the amount of compression that's possible depends on what the movie is like.

LOSSY AND LOSSLESS COMPRESSION

Clever compression methods can reduce the amount of data a lot, and we can still reconstruct the original signal perfectly. This is called **lossless** compression. If we're willing to lose some of the data, we can compress it still further. This is called **lossy** compression, and it means the original signal cannot be completely restored. How lossy depends on how much data is thrown away. For example, it's possible to convert stereo audio to a data stream with a bit rate of only 32kbps – if you're willing to listen to crunchy, hard-as-nails sound. I personally can't stand those artifacts, but many people have grown to like them. For comparison, top-drawer uncompressed digital audio requires a bit rate almost 100 times higher, and many audiophiles insist that even this is inadequate. Many audiophiles also insist that uncompressed audio and compression that's supposed to be lossless sound different. Maybe we still have things to learn.

Digitized video is almost always lossy to a greater or lesser degree. DVDs don't lose much; Youtube loses a great deal for the sake of reasonable download times.

Sometimes, data is lost by accident, as when there's a scratch in a DVD that corrupts more data than the system can correct. In this case, to cover the loss as much as possible, the system takes data that's already been used and repeats it. Or, it looks at the last data before the glitch and the first data after it and estimates what should have been between. This is called **interpolation**.



A common form of lossy compression is the color averaging commonly done in video. As discussed in more detail in chapter 9, the eye's ability to detect fine-grained brightness variations is much better than its ability to detect fine-grained color variations. An image where the brightness information is sharp and the color information is blurry will look sharp to the eye. Knowing this, the people who designed the analog NTSC system way back in the early 1950s made the best use of the available RF spectrum by separating the signal into one brightness and two color signals and transmitting the latter with much narrower bandwidth.

There's a digital version of that, called **chroma subsampling**. In digitized video, there are three signals:

- Y, brightness
- Cr, the difference between Y and the amount of red.
- Cb, the difference between Y and the amount of blue.

Y, Cr and Cb are recombined in different ways to get various shades of brightness with various shades of color. For example, where you have brightness but no red or blue, you get green.

Chroma subsampling divides a video picture into blocks of four adjacent pixels. Each pixel's brightness information is transmitted separately, but the system averages the color over several pixels. By taking full advantage of the limitations of the human eye, and throwing away information the eye won't see, data can be reduced by more than half – and that's before those other, hi-tech methods are applied.

DIGITAL WAVEFORM GENERATION

Many signal generators now use digital technology to generate analog waveforms. There are two obvious ways of doing this: calculate the values in real time, or retrieve them from a lookup table. For any repeating waveform, the lookup table is a much simpler method. Suppose we start with sixteen pre-calculated numbers that correspond to sixteen sample points in a sine wave. We convert those to binary numbers and put them in sixteen memory locations. Then we read them out, one after another, stepping through those sixteen addresses again and again. We can vary the frequency by simply varying the clock rate. Moreover, the voltage will be stable; no need for anything like the Hewlett-Packard light bulb circuit we looked at in EM3.

Of course, we don't need to limit ourselves to sine waves. Each memory address can hold any number, corresponding to any voltage. This has given rise to the **arbitrary waveform generator**, where the user can create any kind of waveform.

DIGITAL NOISE GENERATION

Noise can also be digitally generated. Consider an 8-bit word: there are 256 possible combinations of 1s and 0s. If we create a file with 256 bytes in it, that permits us to write every one of those possible combinations. If we mix up those bytes as they're being written, so they don't follow a pattern, and then read them out in sequence, we can get a square-wave-like pattern that exhibits the two most important characteristics of white noise: a spectrum that contains energy at every frequency within a specified bandwidth, and equal energy across equal frequency increments – e.g. the same energy between 1kHz and 2kHz as between 11kHz and 12kHz.

Eventually the pattern has to repeat, so it isn't really random – it's a fake kind of randomness called **pseudorandom**. For any number of bits, there is a **maximum-length sequence**; the maximum sequence for 8 bits is 255 clocks long. If we want a longer sequence, we just use longer words.

32 bits gives us a sequence that's upwards of 200 billion clocks long. That's long enough to be truly random in every way that matters. The advantage of pseudorandom noise is that its amplitude is completely predictable, which can never be true of actual random noise. Since we don't want to store 200 billion 32-bit words, we use a circuit that creates the noise in real time. If we want pink noise, we can filter pseudorandom noise just as we do with real white noise.

Our ability to convert analog to digital and back has made us masters of our environment to a degree that couldn't be imagined 50 years ago. Not only do we control voltages, but we control time. We can shift it, overlay it, shatter it into microsecond pieces and reassemble it.

In this chapter, I have not tried to go into much depth; that would require a whole book or set of books. But you do have a pretty good mental picture now of the ways analog signals are converted to digital and back to analog, and some of the more common processing techniques.

One place where we use A→D and D→A conversion all the time is in machine control and robotics. A lot can be done with simple on-off control, but a lot more can be done if we have ways to quantize analog values and process them digitally. In the next chapter, we're going to look at digital control of machines, and as you'll see, mixed analog/digital is right at the center of a lot of it.

End of chapter 7

EM4 Chapter 8
MACHINE CONTROL

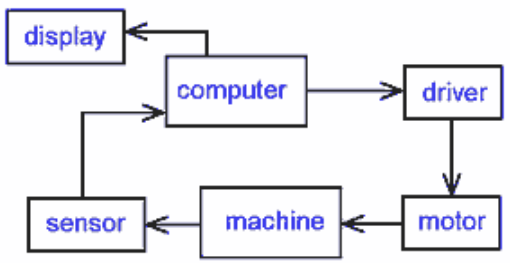
Objectives of this chapter:

- *define CNC*
- *explain why it is necessary to electrically isolate a control computer from the machine it controls*
- *describe various types of machine/computer interfaces and devices*
- *explain why, in many applications, driving transistors must be fully on or off, and spend as little time as possible in transition*
- *define and discuss "open loop," "closed loop" and "servo"*
- *define "overshoot" and how it can get out of control*
- *discuss PID servo loops*
- *discuss set points and hysteresis, and why you might want to have them*
- *explain Gray code and the reason for it*
- *define PLC and discuss variations*
- *explain what DIN rails are and how they are used*
- *discuss ladder logic*
- *discuss variable frequency drives*
- *discuss stepper motors and microstepping*

Machine control is the marriage between computer data and physical machines. It would seem like a shotgun marriage; you couldn't find anything more different than the miniscule electric charge on a flash MOSFET gate and the motor used to lift a bridge. But it turns out that, like Beauty and the Beast, they're made for each other.

COMPUTER NUMERIC CONTROL

Using numbers to control machines is called **numeric control**. When the numbers are supplied by computers, that's **computer numeric control** or **CNC**.



It's not necessary to involve computers; numeric control was once done, and is still occasionally done, with holes punched into a paper tape. The clackety-clack teletype you've seen in old movies was a machine controlled by punched paper tape, with nary a computer in sight. But nowadays, computers are so capable and so cheap that there's usually no reason to do machine control any other way.

BINARY CONTROL

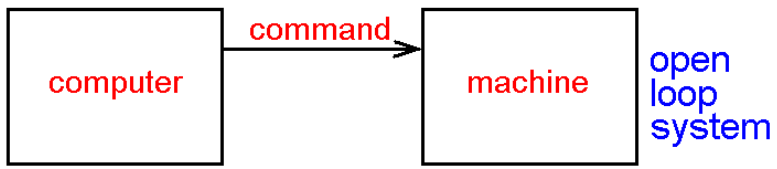
The smallest numerical unit is, of course, the 2-level binary bit, which corresponds to simple on/off control, like operating the remote door latch in an apartment building. A lot of machine control is as simple as that: “1” means open a valve, “0” means close it etc. The skill is in knowing how to choose the right kinds of sensors and actuators, knowing where to mount them, how to connect them and so on.

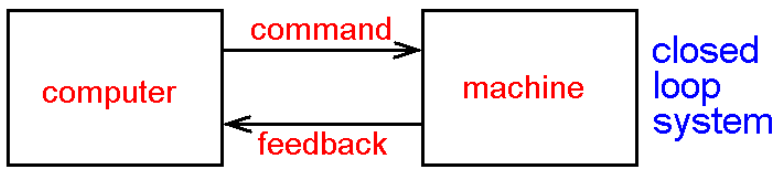
ANALOG CONTROL

Many other applications involve not just “on” and “off,” but “how much” or “how far” – in other words, analog quantities. In almost every case, these quantities are turned into binary numbers either by the CNC computer or an outboard digitizer. In this chapter, when we deal with analog examples, you may assume there's always A-D and D-A lurking in the shadows, and the “thinking” is done in binary.

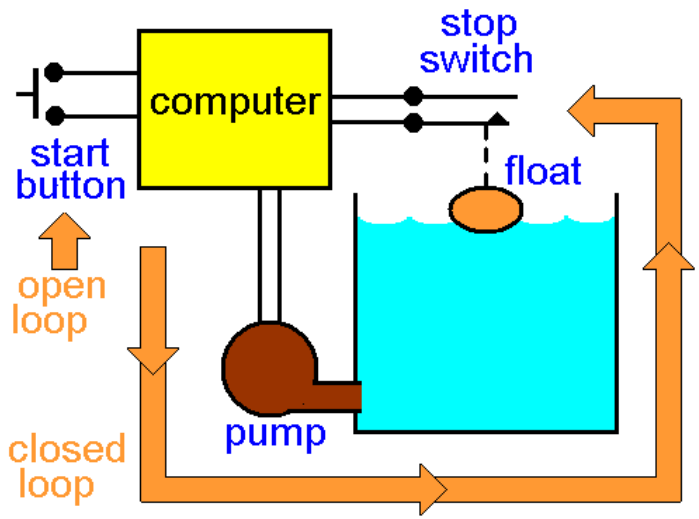
OPEN LOOP AND CLOSED LOOP SYSTEMS

A computer-controlled system may be “open loop” or “closed loop.” An **open loop** system is output-only; the computer issues instructions to the machine but does not receive any kind of “answer.” It may not even know whether there's really a machine connected. A **closed loop** system both sends data to the machine and receives data from it. Since many machines are controlled via RS-232 and similar systems, it wouldn't be unreasonable to compare an open loop system to simplex and closed loop systems to various kinds of duplex.





A system can have both open and closed loops. For example, consider a system where a pushbutton starts a pump that pumps water into a tank, and a float switch stops the pump when the tank is full. The pushbutton represents an open loop; it's a one-way command. The float switch is part of a closed loop; it monitors the condition of the system and reports back to the computer. Another example would be an elevator, where a pushbutton starts the motor and a limit switch stops it when it reaches a certain floor.



Open loop systems are simpler and less costly because the switches and other components that supply return data in a closed system are expensive. There's a big difference between a switch in your clock radio and an industrial switch

that operates reliably, millions of times, in an environment full of oil and dirt.

But many operations cannot be performed without feedback from the machine. Moreover, if anything goes wrong, in an open loop system the computer has no way of knowing. Suppose one part of a machine is moving in a certain direction, and another part of the machine is supposed to move out of the way but has gotten stuck. In an open loop system, they'll just crash. In a closed loop system, the computer wouldn't move the second part until a switch closure told it that the first part had moved.

CNC AND SAFETY

Since I've brought it up, we should talk a bit about the task of programming machine controllers, which you might find yourself doing. It's usually not too difficult to program a machine to do something. The tricky part is anticipating what will happen as if there's a malfunction. In machine applications, when one thing goes wrong, a lot of other things tend to go wrong in quick succession. This can destroy the machine and the stock that the machine is working on, and it can endanger people. A large part of the machine programmer's job is to try and anticipate every possible malfunction and provide code in the program to handle it.

At the very least, there should be a whole lot of “Did the last operation come out OK? → No → Shut down and ring the bell” loops in the program. Remember, computers are stupid, and in an open loop system, blind. Any intelligence that's embodied in the system has to come from the programmer's brain.

MACHINE CONTROL DEVICES

You can take a standard laptop or desktop computer and provide it with interface circuitry to run a machine. A system can be built up of various kinds of control units placed at various points within a machine, all connected to the computer via an RS-232, RS-485 or ethernet bus.

However, many applications don't require that kind of hardware-intensive control. Often, a keyboard or monitor is just something thing to get broken, hardly worth having in a machine that, once programmed, does the same thing for months or years on end. It's much more practical to put the control functions in a module that's made for the purpose, and only plug in the keyboard and monitor when you want to change the program.

Perhaps the most important consideration in computerized machine control is getting data from the machine into the computer and from the computer out to the machine without wrecking the computer. Machines are brawny and computers are fragile. The AC power source that runs a big machine may be many hundreds of volts and have enough spikes and noise on it to reduce the computer to a smoking blob. So, ironically, the CMOS that controls the machine must be carefully isolated from it.

A machine control computer is like Chuck Norris' brain. You've heard the lines: Chuck Norris can fell an oak tree with a swift kick. He doesn't do pushups, he pushes the world down. He once stubbed his toe on a fire hydrant, and the hydrant cried. And so on.

But his brain, like everybody's, is about as tough as custard pudding. It only survives to control his body because it's encased in a skull (reinforced concrete, probably, in his case) and floats in a saltwater bath.

Likewise, the CMOS “brain” of a machine-control computer has to live in a kind of electrical cocoon, and the data that goes in and out must be conditioned so as not to couple the electrical crudeness of big machinery into the CMOS that controls it. This means no part of the machine is ever connected directly to any pin of a CMOS chip. At a minimum, the chip is isolated with build-out resistors; often, relays and optocouplers provide complete electrical isolation.

AVOIDING A HOT CHASSIS

There's another good reason to isolate the computer circuitry from the machine. If the machine is operated directly off the AC power mains, as all big machines are, then connecting the computer directly to the machine would mean connecting it to the power line. The entire computer, and all its connections, and everything plugged into those connections, and everything connected to the things plugged into those connections, would become electrically "hot." This is called a **hot chassis** situation and I bet I don't have to tell you how dangerous it is.

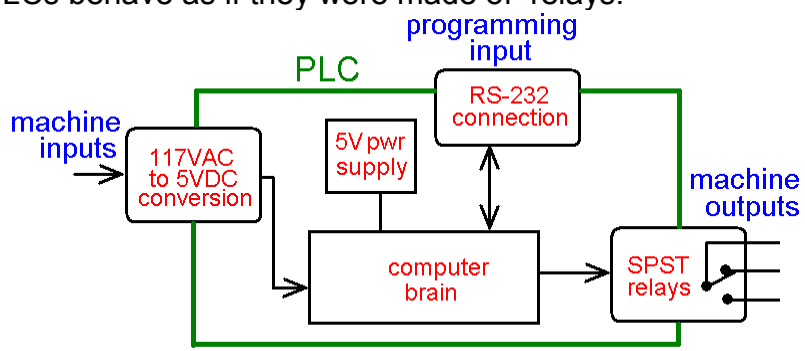
But I'm going to tell you anyway. In fact, I'm going to put it in a frame in big, bold type so you'll know this isn't something to take lightly.

When assembling, testing or troubleshooting a CNC system, accidentally cross-connecting parts of the system people could touch with the AC power line is a constant danger. IT IS YOUR RESPONSIBILITY TO SEE THAT THIS DOES NOT HAPPEN. There is no magic preventative; there is only constant alertness and thinking two steps ahead of every move.

THE PLC

We're going to focus our study on the **programmable logic controller** or **PLC,** a device that combines an MCU with the necessary input and output interface circuitry to run a machine. Since a PLC is intended for industrial applications, it is **hardened**, meaning that it's designed to work in a hostile environment. Its inputs and outputs are conditioned to withstand electrical spikes and similar problems. And, physically, the PLC may be installed in a rugged, oil- and moisture-proof cabinet.

Many PLCs are designed to operate in a 117VAC (or higher) environment, controlling AC machinery and expecting to see AC input signals. From an operational standpoint, these PLCs behave as if they were made of relays.

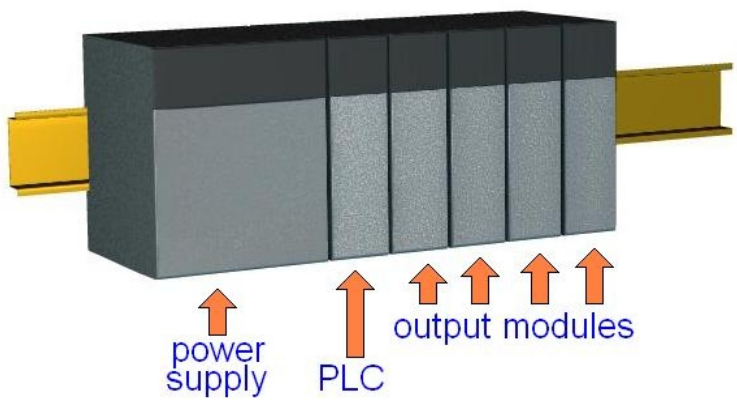


A simple PLC might consist, at a minimum, of:

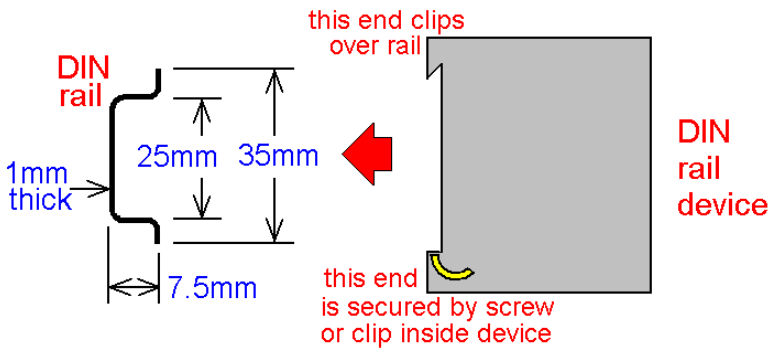
- A number of on-off inputs, probably isolated by opto-isolators.
- An MPU and enough memory to store any program that this class of controller might be expected to execute.
- A number of outputs consisting of relays with SPST or SPDT switch contacts.
- An RS-232, RS-485, Ethernet or similar interface. This may be permanently connected to a central computer that allows a human operator to monitor the machine functions. Or, it may only be connected long enough to load a program. Many PLCs have no display or visual readout beyond a couple of LEDs to indicate that they're awake. The programs must be developed elsewhere and then loaded into the PLCs.

Does such a basic device even qualify as a computer? I would say it does, because it has the computer's most important attribute, the thing that distinguishes a computer from an MCU: a stored program that can be changed. It is, after all, a *programmable* (and reprogrammable) logic controller. This used to mean the majority of its memory was either EPROM, permitting a program to be erased and a new one loaded, or RAM with a battery to render it non-volatile. Today, flash memory provides a more elegant solution.

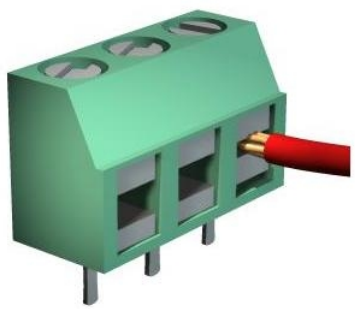
A PLC often takes the form of a module that clips into a **DIN rail**, a standardized mounting bracket that allows modules to be lined up next to each other. Many people refer to the modules as **bricks** because you assemble a system brick by brick, the way you build a house.



The most commonly used DIN rails in the USA have a height of 35mm, a depth of either 7.5mm or 15mm, and a length determined by measuring tape and hacksaw. There are also C-shaped and G-shaped DIN rails that are widely used in Europe.

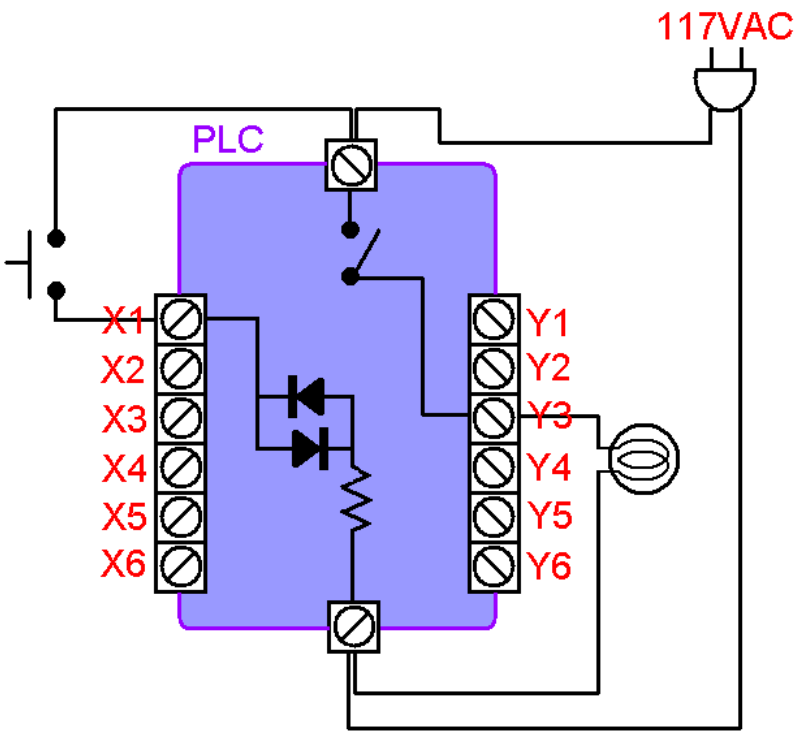


Everything from PLCs to power supplies to circuit breakers to anything you can think of are available in DIN rail bricks. The person putting a machine together can select and interconnect different kinds as necessary to create a complete system. Connections are made either by standard screw terminals, or by screw-type compression devices, often called **Phoenix connectors**. In a Phoenix connector, a wire is inserted into a hole, then a screw is turned to tighten a clamp on the wire. If done properly, this is a good, reliable connection, but there's a certain amount of technique involved. You should practice; working with Phoenix connectors is a basic part of a tech's skill set these days.



The next step in complexity is the PLC with a small picture display. The PLC may include a keyboard, or may have a connector where a keyboard can be connected. This means the programmer can write code right into the PLC, without the need for a separate computer. The display might be a small liquid crystal display like the ones used in wristwatches, or it may be a small full-color LCD monitor. One of the advantages of this kind of PLC is that small program changes can be made right at the machine.

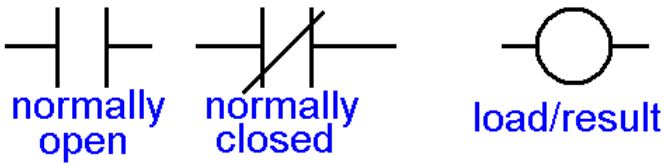
In a PLC, inputs are typically labeled **X** and outputs are labeled **Y**. In the example on the next page, the X inputs are optical isolators. I've shown one input and one output so you can trace current flow. A pushbutton connects one side of the AC line to X input 1; inside the PLC, the optical isolator is connected between the input and the other side of the AC line. The diodes are LEDs, though I haven't shown the arrows; note the 2-way connection. This is a simple way to make DC LEDs work with an AC voltage. On the output side, a relay inside the PLC connects one side of the AC line to Y output 3.



A lamp connected between that output and the other side of the AC line will light when the relay pulls in. Note that the lamp doesn't have to be connected to Y1 for X1 to control it. Which inputs affect which outputs, and how, are determined by the program you load into the PLC.

LADDER LOGIC

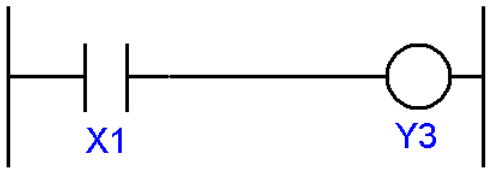
PLC operation is commonly implemented in **ladder logic**, a graphical convention used to represent control functions. Ladder logic can be laid out on paper, but more commonly it's developed on a desktop or laptop computer. In this tutorial we'll deal with just three symbols:



In a ladder logic diagram, a vertical line runs down each side. These represent a frame for logical start-to-finish progressions, which are often completely independent of each other. But they can also be thought of as literal power rails. The symbols that look like capacitors with widely spaced plates represent contacts. The circle can be any kind of load, something that is activated when electrically connected across the rails. In our example PLC, a circle would represent the coil of an output relay whose contacts connect to screw terminals.

Let's develop a ladder-logic diagram of the circuit to the left, where pushing a button lights a lamp. The pushbutton was connected to PLC input X1 and the lamp was connected to PLC output Y3. No reason, just happened that way.

What has to happen inside the PLC? When the X1 input is activated, we need power to be supplied to the Y3 output relay coil. So we go to our laptop computer with the ladder logic development software on it, and we create this:

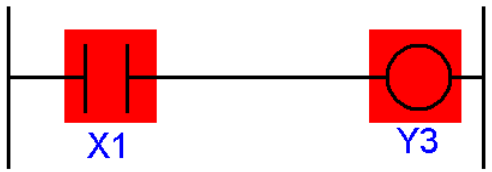


I want to emphasize that the contacts labeled X1 do not represent the pushbutton and the circle labeled Y3 does not represent the lamp. These are virtual components, logic operations that occur *inside the PLC*. The virtual contacts are caused to close when the real pushbutton connected to X1 is pressed. When that happens, the Y3 virtual “load” is activated. In this case the virtual load corresponds to an actual relay coil inside the PLC which closes switch contacts to light the external lamp.

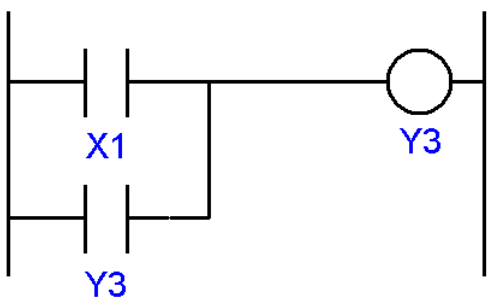
And why does X1 activate Y3? Because we told it to. We wrote a ladder logic program containing that command, and stored it in the PLC's memory.

Before we load that program into a PLC – even before we go out and lay down a Franklin to get a PLC – we can run the program right on our laptop to verify that it works. Ladder logic software almost always includes an emulator, a virtual PLC for testing.

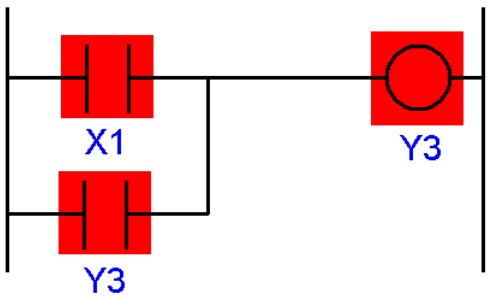
In the emulator, when we activate X1, that connection and Y3 will be highlighted to show that they're active:



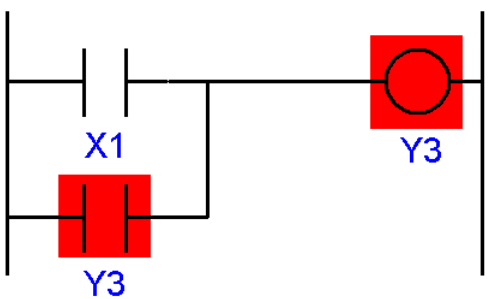
Let's take the next step: making the circuit latch “on” when the button is pushed. To do this, we add a second set of contacts. Again, these are not actual contacts but virtual contacts, and in this case they aren't even wired out to a pushbutton or other device. We'll give them the same label as the output, Y3. In ladder logic, that ties them together so whenever output Y3 is activated, contacts Y3 are also activated.



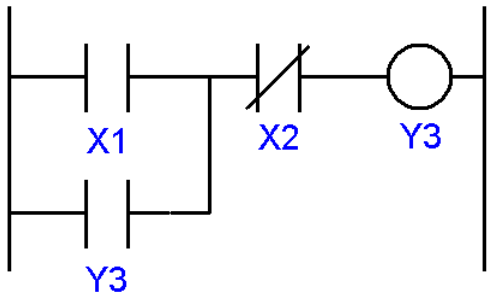
We run the program and push the button. X1 closes, activating the Y3 output and, since it has the same label, the Y3 set of contacts. For as long as the button is held in, the highlighting will look like this:



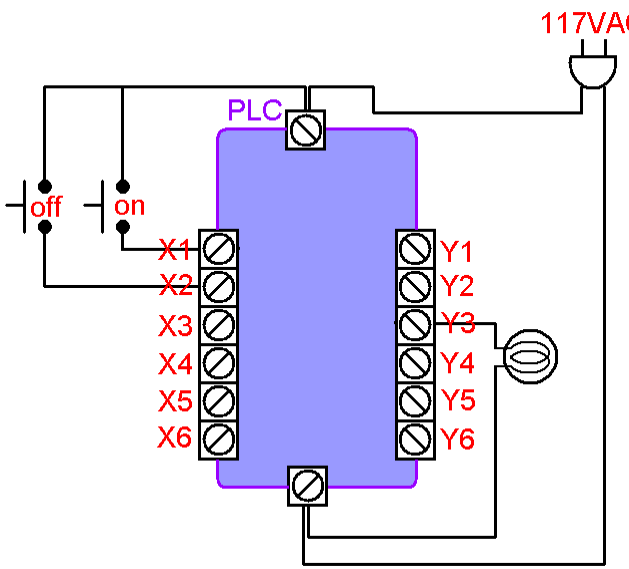
When the button is released, X1 becomes inactive. However, the Y3 contacts are active, thus the Y3 output is active, so that part of the circuit forms a feedback loop that keeps the active status going.



Now, how do we shut the lamp off? We add a second pushbutton that controls a normally-closed function.



In ladder logic it looks like the illustration at left. Note that we're using a *normally open pushbutton* to activate a *normally closed ladder logic function* as shown in the lower diagram. This is not the best idea where dangerous machinery is involved. If the pushbutton connection to X2 is lost, you won't be able to shut the machine off. This isn't just my paranoid self looking for monsters under the bed; it's actually happened to me and to people I know.



If I were designing this system, I'd use a normally closed button and a normally open function for X2. That way, if the “stop” circuit fails, the machine won't start. Annoying, but much better than a machine that won't stop.

The example we've been looking at is so simple that it doesn't require a PLC. A DPST relay can latch on its own contacts. Where the PLC earns its keep is in more complex applications involving dozens to hundreds of interlocking instructions. In a complex logic diagram, there will be so many horizontal lines with closures and circles in them that they begin to resemble the rungs of a ladder; thus “ladder logic.” There are many detailed ladder logic tutorials available online, many for free.

OTHER PROGRAMMING METHODS

Ladder logic is a PLC programming language intended to be simple enough for electricians with no computer training to use. That's its virtue; its limitation is that many complex operations are difficult or impossible to implement in ladder language. Many newer PLCs can now be programmed with Boolean operations or **structured text**, which is how programs are written for MCUs. If you start by learning ladder language, you won't have to throw away everything you know about a particular PLC when you move on to more advanced programming. Most PLCs offer a choice of programming methods.

Next, let's look at PLC inputs and outputs. I've said that they are not direct connections to CMOS logic, so what are they?

BINARY (ON-OFF) CONTROL OUTPUTS

Even sophisticated machines are often based on simple on-off functions. The sophistication comes in the number and variety of functions, their integration and interaction, and in precision timing. Most PLC outputs are designed to control on-off functions.

RELAYS

It's hard to improve on the low-tech old standby, the relay.

- They are simple and well proven, and have been around for a century and a half. The bugs have been worked out.
- They are tough and won't blow up if momentarily overloaded.
- They don't care whether they're switching AC or DC, or what the voltage and current are, as long as their limits aren't exceeded.
- They provide complete electrical isolation between computer and machine.

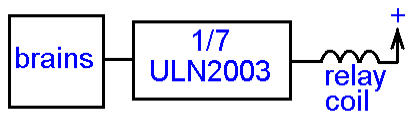
- Controlling a huge machine is a simple matter of selecting a huge relay. If the PLC can't drive the relay coil directly, it can do so indirectly via an intermediate relay.
- A modest inductive flyback when the contacts open won't damage the relay.

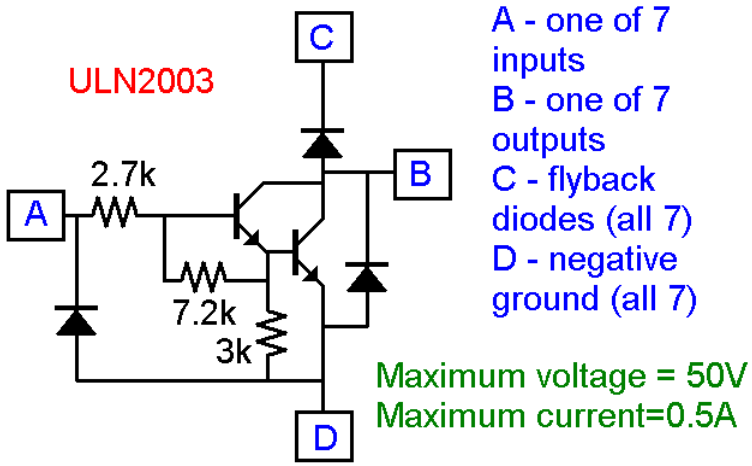
On the downside, relays can get dirty and wear out. They don't operate as quickly as solid-state devices and their contacts bounce. There are also important precautions:

- If the contacts are required to switch excessive current, they can fuse together. This is an extremely dangerous situation because it usually means the machine won't turn off. Any experienced tech could tell you horror stories. Wherever fused contacts would result in an emergency situation, you should choose a relay with contact current ratings far higher than the actual current in the particular circuit. Bear in mind that start current is usually much higher than run current.
- An additional manual shutoff should *always* be provided. This is just good machine control practice, and usually required by law.
- To avoid limiting the applications a PLC can be used for, the manufacturer generally does not supply flyback suppression across the relay contacts. That's your job. Even if a flyback won't damage the equipment, it will radiate a spike of electromagnetic interference that could cause problems.

A given relay will have several different ratings that correspond to different conditions. Generally, contact ratings are lower when switching DC, because of DC's tendency to arc.

A PLC's MOSFET circuitry can't drive a relay coil directly, so the PLC will include a FET or Darlington BJT driver in each Y circuit for that purpose. One of the most popular devices, which you'll also run into in other places, is a ULN2003 chip (or equivalent) containing seven NPN Darlington pulldowns. Note the built-in flyback diodes.



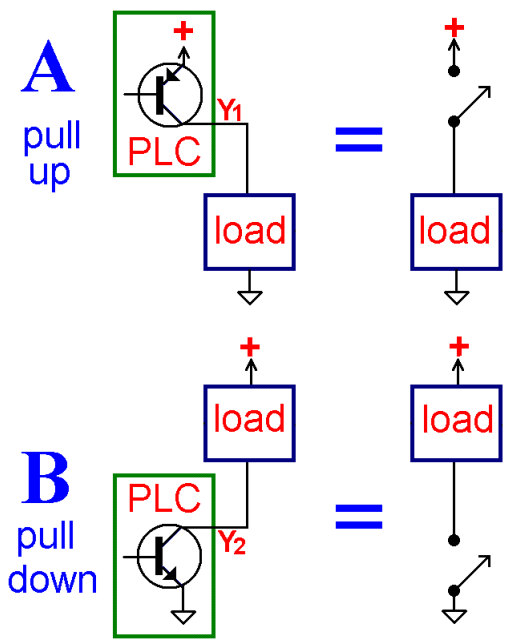


Up to this point we've been looking at PLCs with relay outputs, but probably the majority these days do not use relays and their outputs are not isolated from their internal circuitry. This is especially true of smaller PLCs mounted on DIN rails. Instead, a PLC has outputs that switch either to the +DC rail or ground. If a relay is needed, it's provided in another brick, often at quite a distance from the PLC.

SOLID STATE OUTPUTS

If a PLC has solid-state outputs, they will be transistors that are either "on" (saturated), being equivalent to closed switches, or "off," being equivalent to open switches.

One end of each output transistor will be connected to either the +DC rail as in "A" to the right, or to ground as in "B". We have to know which so we can connect the load properly.



It seems to me that the sensible names for these would be "pull up" and "pull down" respectively. The PNP transistor at "A" pulls its Y output up to +DC; the NPN transistor at "B" pulls its Y output down to ground. These are well-established words with Boolean gates and the like.

But "pull up" and "pull down" aren't the names used by most PLC manufacturers. Allen-Bradley uses the terms **PNP** for a pull-up output and **NPN** for pull-down. Since you've learned how switching transistors work, you can see the logic in these names, though they may be a bit dated now that FETs are taking over the known universe.

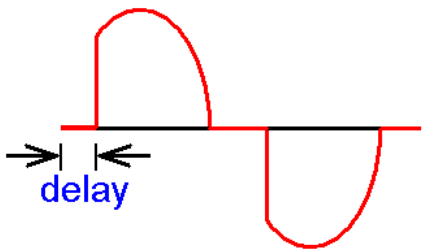
Siemens and Omron use the names **source** for a pull-up and **sink** for a pull-down. You'll remember from EM3 that these names are conventional-current ideas; a source supplies current from the +DC rail to the load and a sink drains current from the load to ground. Some manufacturers use **true/false** designations, as in 1 = true, 0 = false. In any case, once you know which rail the PLC's Y outputs pull to, you connect the other end of the load to the opposite rail.

Let's take a detour and look at some of the devices that a PLC would control, things that actually do the switching in high-voltage, high-power circuits. Then we'll return to PLCs.

TRIACS

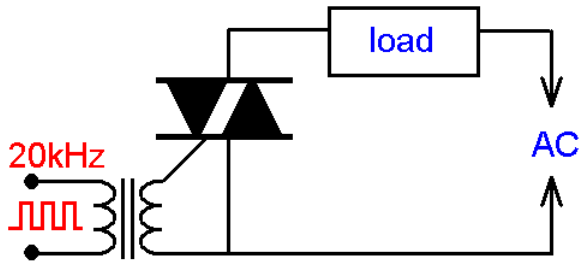
Presumably, I don't have to say much about triacs, since we covered them in EM3. You know that a triac is “off” like an open switch until it receives a trigger pulse at its gate, which turns it “on.” Once on, it will not shut off again until the current flowing through it disappears. This makes the triac unsuitable for DC applications but a natural fit with AC because current reverses (thus momentarily falling to zero) twice each cycle.

When triacs are used in lamp dimmers, there's a time delay between each AC zero-crossing and when the triac turns on. This delay is varied to vary lamp brightness. In simple circuits, the delay is never zero, meaning full RMS AC voltage is never applied to the lamp, even at maximum brightness. When a triac is used as a switch, this is unacceptable. The triac must re-trigger immediately after each zero crossing.

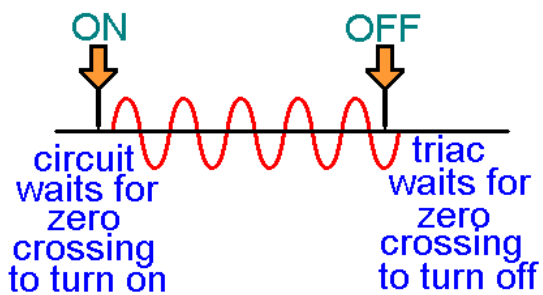


One way of doing this is to continuously feed the gate with a high frequency waveform whenever the triac is supposed to be “on,” so it will be retriggered within, say, 50μSec after a zero crossing. The trigger is coupled through a **pulse transformer**, which is a small, inexpensive transformer that only has enough ferromagnetic material to handle high frequencies. This also takes care of the problem of electrically isolating the triac from the control circuitry.

Note that I didn't show any flyback suppression across the load. Since the triac always shuts off right at the moment when current is zero, there's no flyback!



If we want to add a little sophistication, we can make a circuit that doesn't turn on until the AC voltage reaches a zero crossing. This is called a **zero-crossing switch** and it's a really cool thing because it avoids the nasty spikes when turning something on or off.

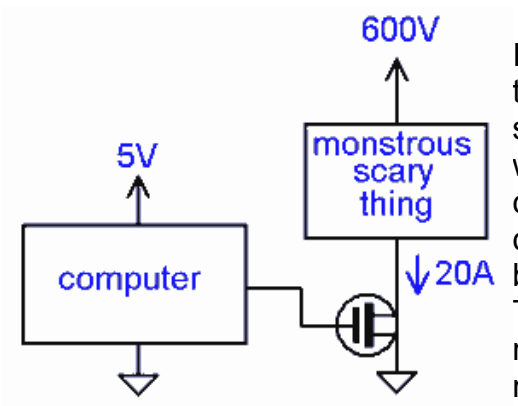


MOSFETS & IGBTS

The large, high-powered things that PLCs control are not always connected directly to the AC power mains. In many cases, the AC is rectified and the high-power stuff runs on DC. That means we can't use SCRs and TRIACs; we have to find something else.

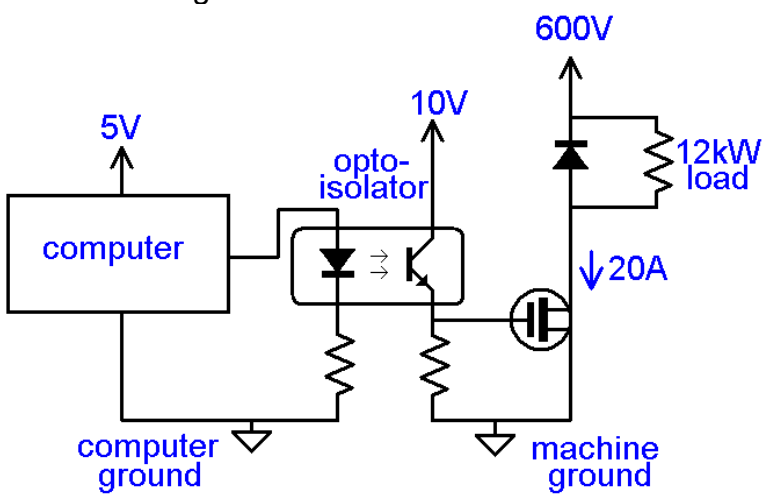
The long-time hands-down best output device in this case is a power MOSFET or an insulated-gate bipolar junction transistor, which is a BJT with a MOSFET's input characteristics.

- They can switch huge amounts of power very quickly.
- Their “on” resistance is often so low it's in the same neighborhood as switch contact resistance.
- They take zero drive current in the “on” or “off” state. The only drive current occurs during switching, due to input capacitance.



In theory, the computer-to-machine interface is as simple as a big MOSFET with its gate directly connected to the computer circuitry. Each bit is either a 1 or a 0. This can become a machine action – turn a motor on or off, set or release a brake, punch a hole and then retract the tool. A few bucks will buy a MOSFET that can switch kilowatts.

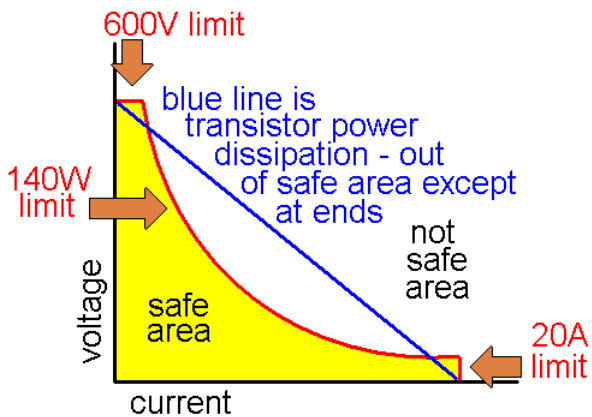
Nothing is that simple, of course. At those power levels, just the capacitive and inductive leakage between the machine and the computer can destroy the computer. We would do well to put an opto-isolator between the computer and the MOSFET, and keep the computer circuitry, including its power supply, separate. This may be necessary anyway, because the FET may require a different drive voltage than what the PLC puts out. And of course, if there's any inductance in the load, we'll need a suppression rectifier to snub the flyback, which would otherwise exceed the FET's breakdown voltage.



DISSIPATION

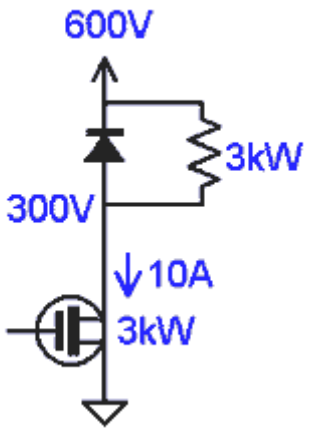
The circuit designer must make sure the FET turns on and off very quickly, because if it ever gets caught handling high voltage *and* high current at the same time, it'll melt in a millisecond. That's one of the basic principles of high-power electronic switching: a transistor may be able to handle a very high voltage in its off state, and a very high current in its on state, but it probably can't handle both at once because its power dissipation limit will be exceeded.

Take, for example, the STW26NM60N, a medium-brawny power MOSFET. The maximum drain-source voltage is 600V, maximum current is 20A, and maximum power dissipation is 140 watts. In the illustration to the right, yellow defines the safe operating area. The horizontal segment at the top is the voltage limit of 600V; the vertical segment at the right is the current limit of 20A, and the curved part is the power limit, VxI, of 140 watts. The blue diagonal line is called the **load line** and represents the power that the MOSFET will dissipate at various points between fully off and fully on.



In theory, we ought to be able to connect a 30-ohm, 12,000 watt furnace heater and use the MOSFET to switch it on and off. When it's off; there's 600V across the MOSFET, but no current; when it's on, there's 20A flowing through it, but very little voltage across it. As long as the transistor is always fully on or fully off we should be okay. (In reality, no designer in their right mind pushes transistors to their theoretical limits.)

But suppose we turn the transistor on only far enough to put 300V across the heater. This will result in 10A of current, causing the heater to dissipate 3,000W. The other 300V will be dropped across the MOSFET which will also dissipate 3,000W. 3kW into a 140W transistor equals smoke and fireworks.



The point is, in high power control circuits a transistor can *switch* much more power than it can *dissipate.* A switching transistor should be completely off or completely on, and spend as little time as possible in the danger zone between.

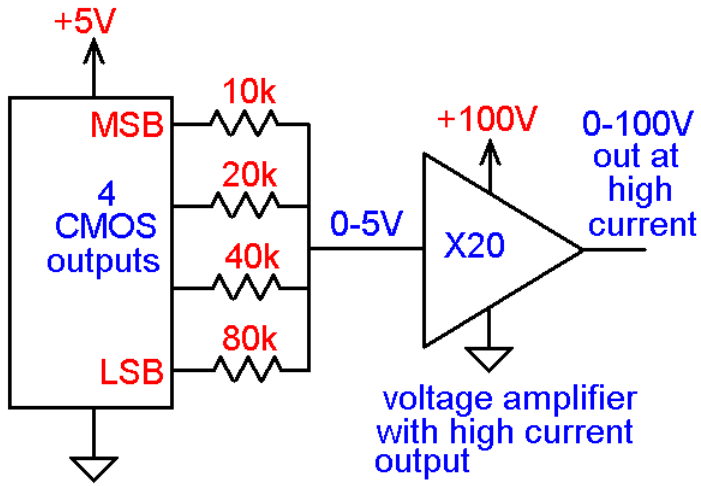
Chapter 8 page 16

DERATING

I've hardly scratched the surface of designing high-power switching circuits. For one thing, the limits listed above only apply when the transistor is at room temperature. When it's running hotter (as it would when pushed hard) the limits must be reduced according to a formula published by the manufacturer. This is called **derating**.

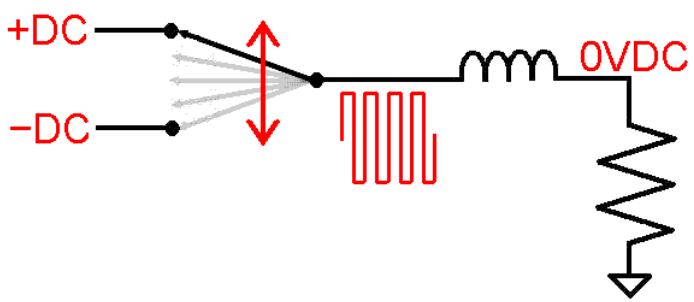
ANALOG CONTROL OUTPUTS

Many machine control applications require more sophisticated control than just on-off. As we saw in chapter 7, binary data can represent analog values. You recall the example of four bits that were turned into one of sixteen analog voltages in a power supply. Suppose we feed that analog voltage into a high-current amplifier. A standard permanent magnet DC motor's speed is roughly proportional to the voltage across it. Those sixteen voltages would, in theory, give us fifteen speeds plus zero speed (off, when the voltage is zero). In practice, at low voltages, a DC motor is unable to overcome friction, so maybe we'd get ten speeds. Or, the voltage could control a current sink that would give us sixteen levels of LED brightness. The circuit above right might be found in an older CNC machine.



The problem with analog control of this kind is low electrical efficiency. If we've got a 100VDC power supply and we want to put 50V across a motor, and it draws 5A at that voltage, then the control transistors have to drop the other 50V. At 5A, they will have to dissipate 250W, which is as much power as is actually delivered to the motor. That's only 50% efficiency, and it entails a huge heat sink and probably a fan. A lot of controlling is still done that way, because a manufacturer isn't going to spend $100,000 for a new machine as long as the old one is working. But nowadays, the class-D switching technology we studied in EM3 gives us a more efficient way to control machines.

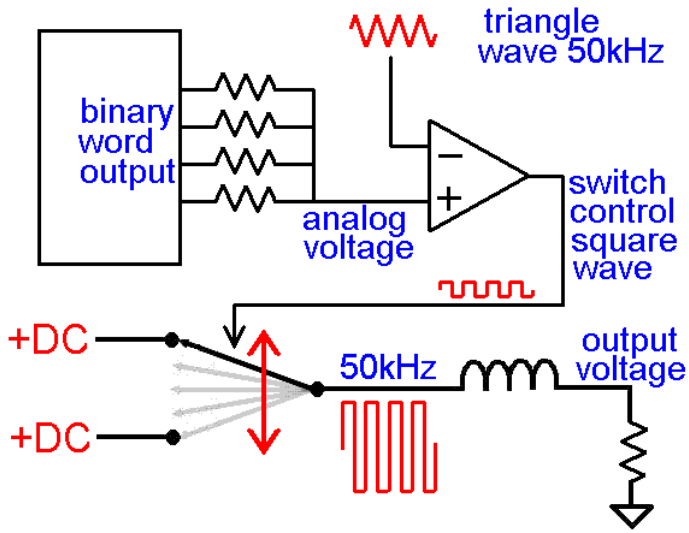
Quick review: if we have a positive and a negative supply, and we connect a SPDT switch that rapidly alternates between them, we'll get a square wave. If we then pass the square wave through an inductor, we'll get a DC voltage that depends on the duty cycle of the switch. A 50% duty cycle will result in a zero-volt output. If the duty cycle is changed, we'll get a corresponding change in the output voltage. We can even generate a low-frequency sine wave by varying the duty cycle in a sine-wave fashion.



In amplifiers, this is usually called class-D operation. In control circuitry, it's called **pulse-width modulation** or **PWM**. In theory, it should be possible to reach 100% efficiency, where every watt that's drawn from the supply reaches the load and none is burned up in the control circuitry. In practice, 90% efficiency or better is usually realistic.

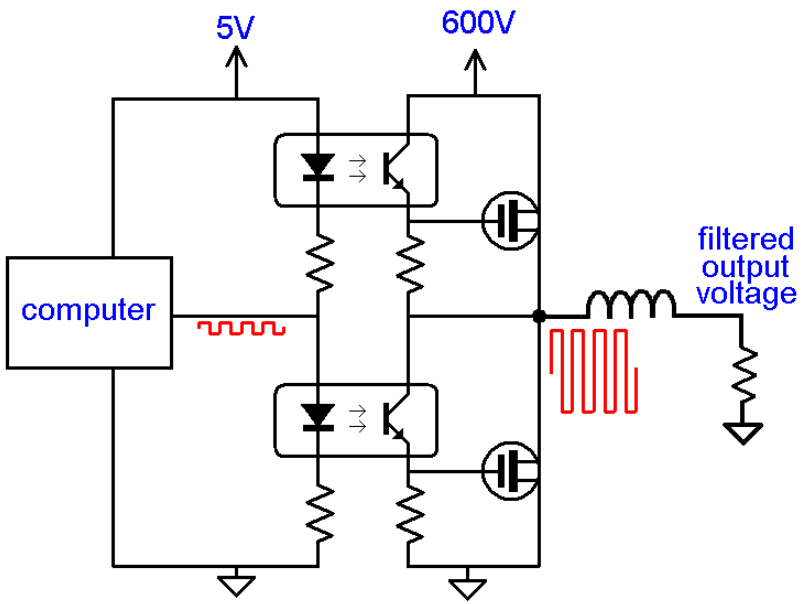
Binary data is converted to PWM in two steps. First, a D-A converter changes binary data into a DC or AC voltage.

Second, the voltage is converted to PWM. One simple way of doing this is to feed the voltage into one input of a comparator and a high-frequency triangle wave into the other as shown below.

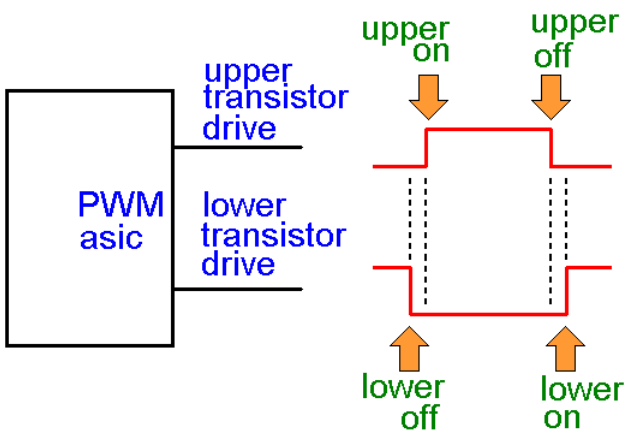


The analog part of this process can be bypassed by using an equation or lookup table to convert directly from binary to a PWM duty cycle percentage.

Many modern MCUs have built-in PWM outputs, so all that is necessary is to take the small MCU output and amplify it as needed to drive a heavy load. If the frequency isn't too high, optocouplers can be used for isolation. The circuit below illustrates the concept, but is not an actual circuit you'd find in equipment. For one thing, you can't put 600V across the transistor of a real-world optocoupler, you'll blow it up.



A potential problem with a circuit like the one above is that transistors take longer to turn off than on, so there is a moment at each transition when both are on. This is the "shoot-through" mentioned in earlier chapters, and it's disastrous at kilowatt power levels. Application-specific PWM chips usually have separate drive outputs for upper and lower power transistors, with independently programmable on and off times. This permits one transistor to start turning off before the other turns on, preventing shoot-through.



Most PLCs do not have these circuits built in. (For one thing, most PLC's don't have analog outputs.) If you're building up a system of bricks on a DIN rail, and you need, say, a PWM motor drive, that will likely be a separate brick controlled by the PLC – or you'll be using an entirely different kind of CNC system.

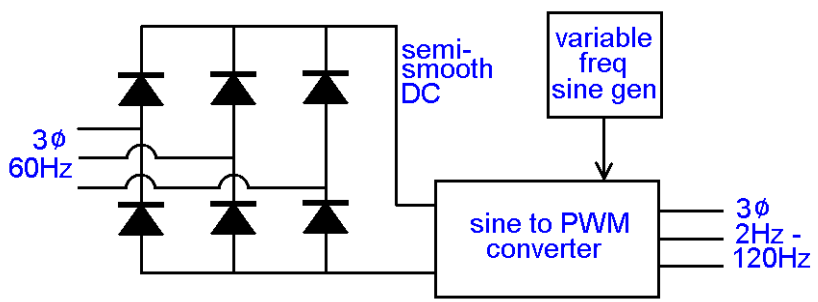
VARIABLE FREQUENCY DRIVES

One particular kind of analog output unit deserves special mention: the **variable speed drive** or **VFD** which, instead of generating a variable DC output voltage, generates a variable AC output frequency.

For many years, variable-speed motors were almost always DC motors, because controlling an AC motor's speed is difficult. An AC motor's natural operating speed is governed by the power line frequency. If that frequency is 60Hz, an AC motor will want to turn at 3600RPM, or a submultiple like 1800, 1200 or 900RPM. Motors can be specially built to permit speed control, e.g. a 3-speed window fan, but these tend to be inefficient and the speed control tends to be crude. Thus designers of servo-controlled machines have long been stuck with the inconvenient necessity of using DC motors and providing a large AC-to-DC converting power supply.

But technology has recently come to the rescue in the form of MOSFETS and IGBTs that are brawny enough to handle the power for multiple-horsepower motors, and fast enough to do PWM with low losses. By combining low-maintenance AC induction motors and VFDs, we can now build cost-effective variable speed machines using AC motors. And, as an added bonus, we've got better speed control. A DC motor's speed is proportional to voltage, but only very roughly, especially if the mechanical load on the motor varies. But with an AC motor, the very frequency-to-speed connection that gave us so much trouble before now benefits us: if we know the frequency we're feeding the motor, we know its speed within a few percent. (If we're using a synchronous motor, which we'll talk about in EM5, we know the speed exactly.) This often eliminates the need for a separate sensor to measure speed.

In its most basic form, a VFD simply rectifies the AC, turning it into DC, then generates a new sine wave at a different frequency. If we start with a 3-phase source and use six diodes, the DC is smooth enough not to require further filtering. The VFD can begin at 2Hz and low voltage, and then ramp the frequency and voltage up, allowing the motor to start gently and avoiding the usual starting overcurrent. At the other end of the scale, a VFD allows motors to be run faster than their rated speed. Most motors will work up to 130% of their rated speed, and some will run at 200%.



VFDs are showing up all over the place; you can hardly find a new washing machine that doesn't have one. The benefit, often, is energy savings. For example, the power required to spin a fan is roughly proportional to the *cube* of its RPM. If you only need maximum airflow occasionally, and can cut the speed in half the rest of the time, you save almost 80% of the electricity.

Many VFDs can be computer controlled, which brings us back to PLCs. We've talked about outputs and things you connect to them; now let's talk about inputs.

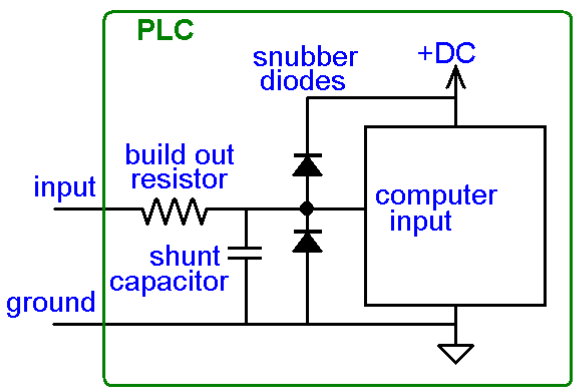
BINARY (ON-OFF) CONTROL INPUTS

PLC inputs are basically of two kinds: those designed to work with comparatively low DC voltages, and those designed for 117VAC (and higher) voltages. The lower-voltage variety may not be isolated from the computer circuitry; the power line voltage variety, of course, must be.

NON-ISOLATED INPUTS

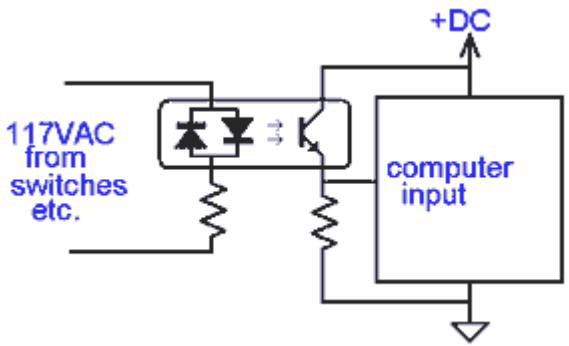
In a typical non-isolated system, the PLC might run on +24V. This same voltage can supply the switches that feed signals to the X inputs. Despite the fact that everything is running off a low-voltage DC supply and grounded together, input conditioning is still needed. The most effective measures combine the following:

- A build-out resistor, to limit current.
- A snubber diode between the input and the computer's ground, to insure that the input can't be pulled more negative than the ground rail.
- A snubber diode between the input and the computer's +DC rail, to insure that the input can't be pulled higher than the supply rail.
- A shunt capacitor to short out hot, fast spikes. This has to be chosen carefully, and cannot be used in some situations, because it also shunts high-speed data.



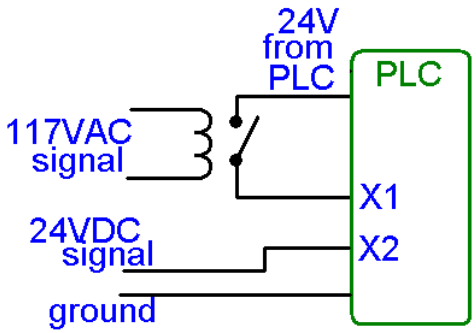
You can pretty much assume that a commercially built PLC has as much conditioning circuitry as necessary to protect the PLC. However, you may still have to deal with noise problems, as we'll see.

ISOLATED INPUTS



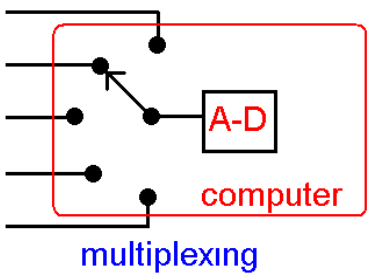
Where control input signals are derived from the AC power line, full isolation is naturally required. An optocoupler with two LEDs connected in opposition easily provides this function. There are two reasons the dual LEDs are required. First, an AC input should sense a voltage of either polarity. Second, with only one LED, the full AC reverse voltage would appear across it during the half-cycle when it's not conducting, which would destroy it.

This brings up yet another reminder about the danger of a hot chassis. It isn't unusual to want to connect different voltages to the different X inputs of a PLC. Nothing wrong with that, in principle. But suppose one of those comes from the 117VAC power line, and another is a DC signal that's connected into the PLC ground circuit. There is no way you can connect both up directly without creating a hot-chassis situation. The 117V control signal will have to be isolated using a “repeater” relay. I'm not familiar with every PLC, so there may be an exception: a PLC where *each* of the X inputs is completely isolated. General principle: a tech who works with PLCs should become completely familiar with what's going on inside. That's one of the things that separates an ace tech from a duffer.



ANALOG CONTROL INPUTS

The obvious way to convert an analog input voltage into the digital form needed by a controller is to feed the voltage into an A-D converter. If we're not quantizing fast-changing data, a single A-D can be switched among several inputs, quantizing the voltage on each in turn. This is called **multiplexing** (**MUX** for short) and it permits a single MCU to monitor analog voltages at many points in a machine. Some PLCs have this function built in, but usually you'll need another brick with its own MCU to do this.



Chapter 8 page 22

A typical MCU's converter will have a bit depth of 8, 10 or 12 bits; the latter can quantize a voltage to an accuracy of one part in 4,096.

If we just need to know whether a voltage is higher or lower than a reference voltage, we can use a comparator. Again, many MCUs have built-in comparators. A comparator can also be used to make a slow A-D, by using one of the computer's timers to measure how long it takes to charge a capacitor up to a reference voltage. In chapter 7 we saw that digital voltmeters are usually built that way.

Summary: PLC inputs are usually binary (on-off). Some PLCs have analog inputs, or a separate brick with analog input and digital output may be used. If the PLC is a long distance from the machine being controlled, putting the analog sensing function at the machine and sending the data digitally may help keep noise from corrupting it.

Now let's look at some of the sensing devices that may be connected to the PLC's inputs.

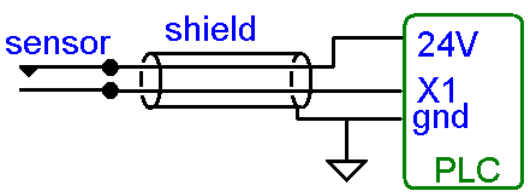
ON-OFF INPUT DEVICES

There are so many standard devices, and so many more that are unique to their varied applications, that it's hard to know where to start. If you're laying out a new system, one of the first decisions you'll need to make is whether the sensing devices will have their own power source – e.g. a switch connected to the AC mains that sends power to an X input when it's closed, via an isolating relay – or whether they'll derive power from the PLC's own DC supply.

PRECAUTIONS

A PLC usually has low-current, high-impedance X inputs so it can be used with a wide variety of sensing devices. This can cause problems. In an industrial setting, the cable between the sensing device and the PLC can pick up enough noise to give false readings, even if the sensing device is as simple as a pair of switch contacts. Here are some rules of thumb to keep you out of trouble.

- Every PLC I've ever seen has a ground terminal. Be sure it's Earth-grounded. Conduit ground is usually an acceptable Earth ground.
- If the sensing device is more than a few feet from the PLC, do not ground any part of its circuit at the machine. Use a twisted pair and return to ground (or hot) at the PLC.



- If it's a long cable run through a hostile electrical environment, use shielded cable. Ground the shield at the PLC end only.
- If the sensor requires separate DC, ground and signal connections, use 3 twisted wires. If you allow an open loop between conductors, that's an antenna.
- Reread the cabling chapter of the EM2 study guide.

All of that is fine in theory; the inconvenient fact is, electrical codes often require grounding the sensing device at the machine as well as the PLC. This creates a ground loop, which may mess everything up. If the sensing device is something simple like a switch, you may have an out: it may be enclosed in a case which can be locally grounded without grounding the switch inside, as in that limit switch I showed you before. Otherwise, you may be stuck running conduit from the controller to the machine, just to enclose the wire for that one switch. Bummer.

The brawniest contacts don't necessarily make the best ones. High-power switches and relays often depend on the high current that flows through them to keep the contacts clean. If you connect a switch intended for a high-power application to a PLC's X input, you may find that it's unreliable. Switches are broadly divided into **power** switches and **signal** or **control** switches. If you have a choice, you want the signal ones. BTW, a circuit that only handles a low-level signal, not power, is often called a **dry** circuit.

REED SWITCHES

A reed switch is a magnetically activated device consisting of a pair of contacts enclosed in a sealed glass tube. The contacts are mounted on beams made of ferromagnetic material. When you bring a magnet nearby, the beams become magnetized by induction, pull toward each other, and the contacts close. The most familiar example is the switch mounted on a door or window frame and used to set off a doorbell or alarm.

Reed switches provide a reliable electrically isolated way of switching dry circuits. Because the tube is sealed, the contacts never get dirty. They can operate millions of times before they wear out. If a coil of wire is wound around the tube, the switch becomes a **reed relay.** Their main limitations are that they can only handle a few milliamps and they can very easily be broken by twisting or bending the leads.

They're also not very precise or repeatable, and they have considerable hysteresis, meaning that after the magnet is moved so as to turn the switch on, it has to be moved back a ways for the switch to turn off.

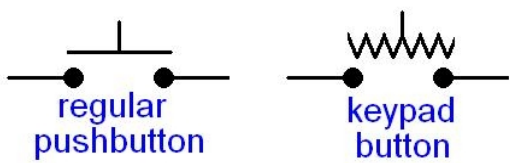
HALL EFFECT SWITCHES

A Hall effect switch looks like a small plastic transistor, but it senses the presence and orientation of a magnetic field. Hall switches have no moving parts and do not wear out.

Magnetic switches have an important virtue: their operation is not affected by dirt, grease, or gunk.

KEYBOARD SWITCHES

If you ever have to interface a keyboard or keypad to a PLC, be aware that keyboard switches nowadays usually consist of conductive plastic or conductive rubber actuators that short across PC board pads. The word “short” needs to be qualified, because the conductive parts' resistance is nowhere near zero. If you find that operation is unreliable, measure a key's “on” resistance. Then look at the PLC specs to find out if the X-input voltage and current requirements are such that a switch with significant resistance could cause a problem.



PHOTOCELLS

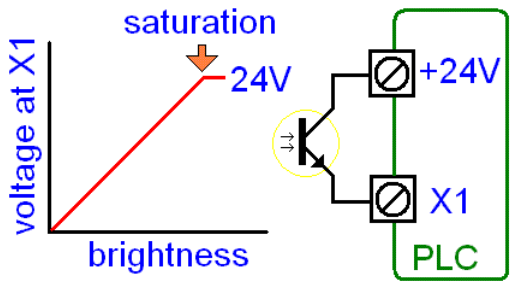
The nonzero resistance comment also applies to photocells, specifically photoresistive devices, whose resistance changes in the presence of light. Also, photocells are not very fast. Their resistance is low when illuminated and high when dark. Since they're resistors, current can flow either way and they can conduct AC.

PHOTOTRANSISTORS

All solid-state devices are at least slightly light-sensitive, which is why so many of them are encased in black plastic. A phototransistor is simply a transistor in a transparent case, so light can reach it, with characteristics optimized for light detection. There may or may not be an available base connection.

Phototransistors are not especially fast, but they're more than fast enough for most optical sensing duties in machine control applications. **Photodarlingtons** are more sensitive but slower.

Current through a phototransistor is proportional to light intensity. This means, if you want a definite "on" with almost no voltage drop across the transistor, you just have to make the light bright enough to saturate the transistor.



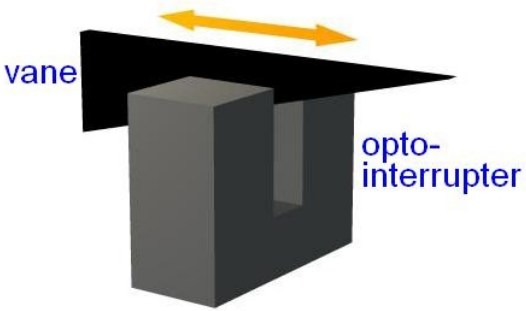
Optical sensors are not a good choice in dirty environments, because the light path can become blocked. More information on optical sensors can be found in the EM3 study guide, chapter 10.

ANALOG SENSING DEVICES

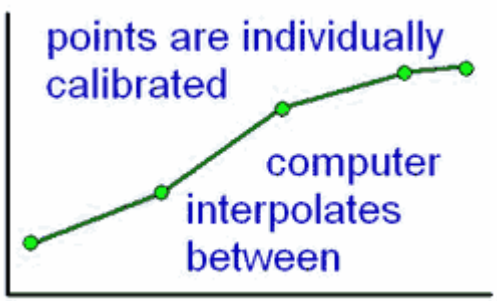
Some PLCs and many other kinds of digital control units feature analog inputs which permit measurement of varying voltages. The proper sensor depends, of course, on what we're measuring. For example, temperature can be measured using a thermistor, which is a 2-lead device whose resistance varies with temperature. Thermistors are fairly linear over a limited temperature range, but they differ from one to the next, so they need to be calibrated. If more precise temperature sensing is needed, there are ICs which combine sensing and output conditioning, and come pre-calibrated to a very tight tolerance. Caution: you have to be sure that the PLC doesn't put enough current through the sensor to heat it. In many cases, you'll wind up buying a complete system consisting of the thermal sensor and the brick it's connected to.

If we're measuring position, the most obvious sensor is of course the potentiometer. If we set up the pot so the end positions of the wiper correspond to the limit positions of the machine, we can tell where the machine is by the voltage coming from the pot. However, this is a poor solution. Pots wear, and the output changes over time. If the wiper gets dirty, all contact is lost. A better solution is to use an optical or magnetic sensor, which does not change over time.

A wedge-shaped vane between a light source and a photocell or phototransistor can vary brightness on the sensor linearly with position. If we want to sense rotational angle, we can use a disc whose transparency varies. If we want to use magnetism, Hall effect sensors are available whose voltage varies linearly with magnetic field strength.



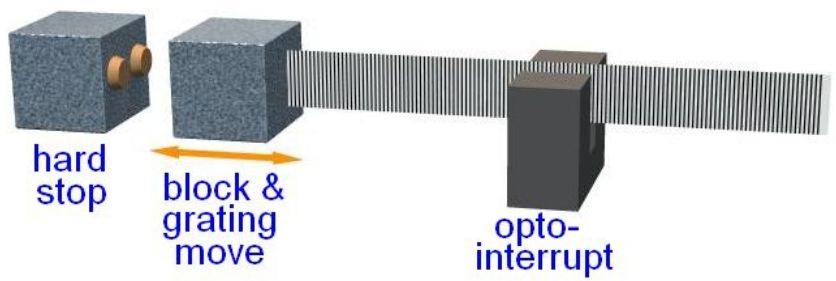
These methods don't yield accurate position indication, but accuracy can be obtained through **calibration**. If you've set up a large TV, you know about this process. Reference points are established at various brightness levels between darkness and full brightness, and the color is balanced at each point. The TV then estimates values between the points, correcting for nonlinearity so that, for example, grey doesn't turn to blue in dark parts of the picture. The same method can be used to get accurate results from optical and magnetic sensors. The sensors are calibrated at various points and the computer fills in the information between. (This is an example of **interpolation**.) It doesn't matter if the sensors are nonlinear as long as they are stable. Many programs include easy calibration functions: set the machine to a certain position, push a button, the computer writes a value into memory.



RELATIVE POSITION SENSING

Another way to get an accurate position value is to use a sensing system that outputs discrete steps. The computer can calculate position by counting increments as the machine moves.

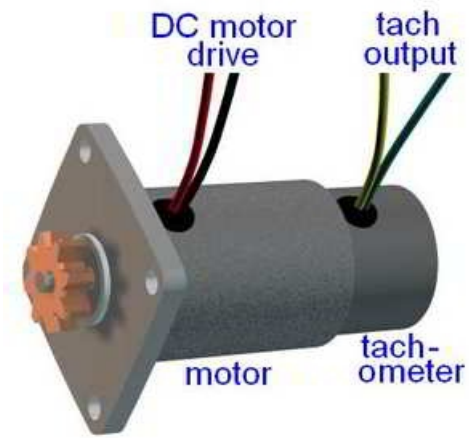
In the illustration below, the movable block representing a machine is attached to a grating strip marked with fine vertical lines. The block and grating are first moved to the left until the block rams up against the hard stop or closes a pair of switch contacts.



This is “home base,” location 00000000. The machine now knows where it is, and can move to another specific position by **indexing** from the known position. As it moves to the right, the grating strip passes through an opto-interrupter which counts vertical lines in the grating. The accumulated count at any moment represents the position of the machine. If you take apart a printer, you're likely to see this system in reverse: the grating stays in one place, while the print head with an opto-interrupter attached moves back and forth.

When we talked about the computer mouse, we discussed the use of two opto-interrupters working in quadrature to determine which way the mouse was moving. Direction sensing is unnecessary when the controller is driving the machine because, after all, it's telling the machine which way to move, so it already knows the direction.

If the machine is gear-driven, there will be no slippage as there would be with belts and pulleys. The gear ratios will impose a fixed relationship between the motor rotation and the position of the machine. In this case we can attach the grating counter to the motor, simplifying construction and keeping the sensor out of harm's way. A device that puts out a certain number of pulses with each motor rotation is called a **tachometer**. In this case, the operating principle is often not optical but magnetic, using Hall effect sensors.

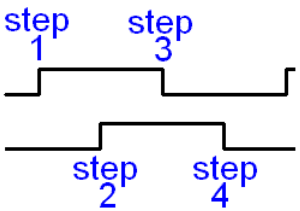


When a motor is operated within a closed loop, with its speed monitored by a tachometer, any error between actual speed and desired speed can be detected and corrected by adjusting the drive voltage (or, in the case of an AC motor powered by a VFD, by adjusting the drive frequency). A motor used inside a feedback loop is called a **servomotor** and the system is a **servo loop**. A familiar example is a dejay's turntables, which must rotate at exactly $33^{1/3}$ RPM. The turntables are driven by motors controlled by magnetic tachometers.

STEPPER MOTORS

We're going to study motors in detail in EM5, but I want to mention a special kind of motor called a **stepper** whose rotation is divided into discrete steps with magnetic detents, “X” number of degrees per step. For example, a particular motor might divide a 360^{O} revolution into 120 steps, turning 3^{O} with each step. If this motor is driven with discrete pulses from a controller, its position at any given moment corresponds to the number of pulses it has received.

Thus the stepper gives us a way of translating number of pulses directly into position.



A typical stepper has two coils and is stepped by two square waves 90 degrees apart.

In theory, a system using a stepper can be an open-loop system. Feedback is not required to know the motor's position. In practice, this doesn't always work, because it is possible for the motor to "slip" and miss or overshoot a step, and without feedback the controller has no way of knowing. There are three ways of dealing with this.

If damage from a slipped step is not serious, the user may choose to ignore the problem. For example, if a stepper is being used to spool off identical lengths of wire, and one time in ten thousand it slips, that means 0.01% of the product must be discarded. Perhaps that's less expensive than developing a more sophisticated control system.

The second way is to return the stepper back to the hard stop position after every operation. This will not harm a stepper motor motor if done right; the motor will just slip a magnetic detent or two.

The third way, of course, is to provide the system with a full closed feedback loop.

Steppers are an ideal fit for many machine applications because they're controlled digitally, in discrete steps, which makes for an easy controller interface, but produce a semi-analog output rotation. I say "semi-analog" because the positional resolution is limited by the number of steps per revolution. Resolution can be increased by gearing the motor down; for example, a 3^{O} motor with a 10:1 gear reducer becomes a 0.3^{O} motor.

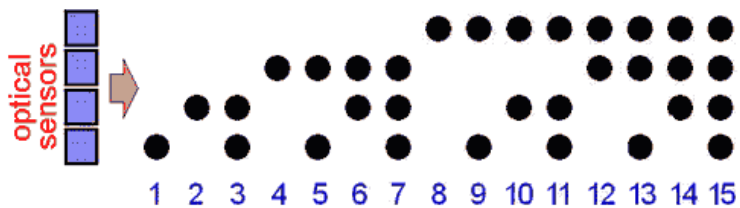
Another way of achieving finer control is to partially activate adjacent coils with lower-than-normal voltages. This results in an in-between magnetic detent, meaning that the motor takes a position between steps. This is called **microstepping**. Most printers use it to obtain very precise positioning of the print heads.

With some steppers, if the coils are fed with sine waves, rotation becomes smooth, like any other motor, but with precisely known position at all times.

ABSOLUTE POSITION SENSING

Absolute position can be determined if, instead of a single optical sensor counting lines, we have a binary word's worth of sensors and a strip with binary code on it, which may be represented by black dots on a white background.

We don't need much in the way of interface circuitry; the outputs of the sensors form a digital word that can go straight into the controller.



We do have a problem, though, because in an optical sensing system the transitions between 1s and 0s are not precise. Suppose 1s are represented by black, and we're moving from position 1001 to position 1010. If the illumination is a little dim, or the black dots are large, there will be a moment when the two least significant bits will seem to be 1, so the binary count will go 1001 → 1011 → 1010. When the controller sees 1011, it will think it has overshot the target and reverse direction.

This problem is solved by using a modified binary counting system called **Gray code**, also called **cyclic code**, invented at Bell Labs by Frank Gray. In Gray code, the 16 possible binary patterns are shuffled so that, between any two adjacent numbers, only one bit changes at a time. Note also that the 15_{10} state can roll over to the 0 state with only one bit change.

GRAY CODE

0000 0	1100 8
0001 1	1101 9
0011 2	1111 10
0010 3	1110 11
0110 4	1010 12
0111 5	1011 13
0101 6	1001 14
0100 7	1000 15

This isn't limited to the 15 values shown. We can use as many bits as we need for a number of any size, and the basic Gray code rule remains true: from any number to the next number, one bit changes.

Learning to count in Gray code is easier if you note that the LSB follows a pattern of twos; that is, two 1s, then two 0s. The next bit follows a pattern of 4s, then 8s, then 16s.

In a computer or controller, the easiest way to deal with a Gray code is via a lookup table.

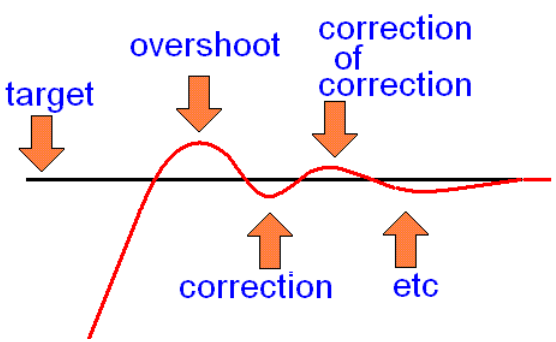
LOOP STABILITY AND DAMPING

One of the things a closed-loop control system has to deal with is the fact that machines don't start and stop instantly. **Inertia** and **momentum** are really the same thing; inertia causes a machine to start slowly, while momentum causes a moving machine to stop slowly. Both are caused by mass of the moving parts, which resists a quick change in speed or direction.

If you shut a machine off when it reaches the point you want it to reach, it will keep going past the desired stopping point due to momentum. This is called **overshoot**.

These effects don't just apply to motion. For example, if you're heating a room with a hot-water radiator, heating doesn't stop when the room reaches the desired temperature. Even after the thermostat turns the water off, you've got that big hunk of hot metal sitting there, and it takes a while for it to cool down, during which time the room gets warmer than you want it to. Another form of overshoot.

In a closed loop situation where the system is controlling a motor, the servo detects the overshoot and tries to correct by applying a reverse voltage. This in turn causes overshoot in the other direction. It may take several cycles of overshoot to finally settle at the correct point. You probably experienced this yourself, when you were learning to drive, before you learned how to steer without oversteering.



In the worst-case scenario, the servo system so overcorrects that each overshoot is larger than the last one, and the system goes into self-sustaining oscillation. This is like someone who's driving on ice or wet, oily asphalt, begins to skid, overcorrects, and goes into a spin.

Loop stability is only an issue in a closed-loop system, because it's the result of feedback. Stability depends primarily on three factors:

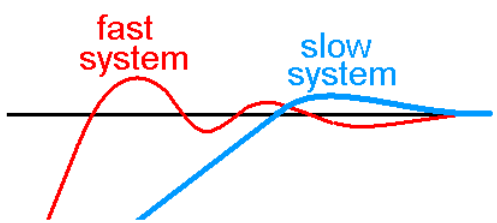
- **Mass** of the system
- **Loop gain** of the servo loop, which means how much correction is applied per unit of of error
- **Velocity** or **speed** of the moving system

The larger the mass, and the greater the speed, the greater the tendency to overshoot. The larger the loop gain, the greater the tendency to overcorrect.

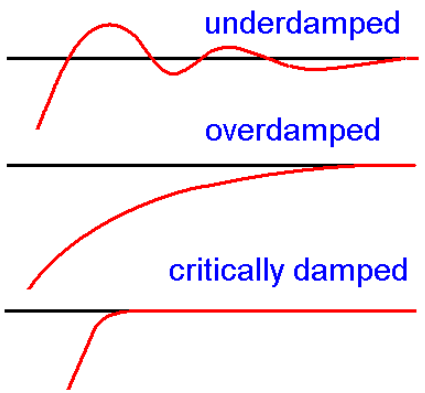
You would think the higher the loop gain, the greater the negative feedback, and the better the error correction. That is true in a system that responds instantly to signals from the servo. But when there's momentum involved, the servo isn't responding to what's happening now, but what happened a little while ago.

Think of it this way: in system with a loop gain of 0.8x, one volt of error results in 0.8V of correction, leaving 0.2V of error. But in a system with loop gain of 3, one volt of error results in 3V of correction, leaving 2V of error. This in turn results in 6V of correction, leaving 4V of error, and so on until the system is completely out of control. If there were no loop delay, overcorrection would self-cancel as soon as it occurs. But when there's momentum, thus a delay between when the servo puts out a signal and the system responds to it, correction comes around the loop too late.

The problem can be solved – we can have a servo loop that's both high-gain and stable – if we make the machine move so slowly that it can't get ahead of the servo loop, and any overshoot is small. Of course, this means the machine takes longer to do whatever it's doing. But when the machine is working by itself under computer control, without a human operator, there's no cost penalty if the machine takes two minutes to do a one-minute job unless you really need maximum throughput.



It's useful to borrow the concept of **damping** from our EM3 study of filters. A high loop-gain, jumpy, overshooting system is **underdamped**. A system that is stable because it's slow, or because loop gain is low, is **overdamped**. Where maximum throughput is desired, what we want is a **critically damped** system – one which reaches its target as quickly as possible with little or no overshoot. This results from a careful balancing of momentum, speed and loop gain. This is something you'll study in detail if your career takes you into machine control, but if you've gone through the EM3 study guide, you already encountered the purely electronic version of critical damping when we were talking about filters.



If you've got the idea that there's an unavoidable 3-way tradeoff among mass, speed and precision that limits what can be achieved, you're almost right. Up to this point, we've been talking only about servo loops that are linear and proportional, like Ohm's Law. What if we throw in a little inductance and capacitance? Or, to put it in servo terms, what if we look not just at where the machine is, but also where it's been and how fast its movement is changing?

PID CONTROL

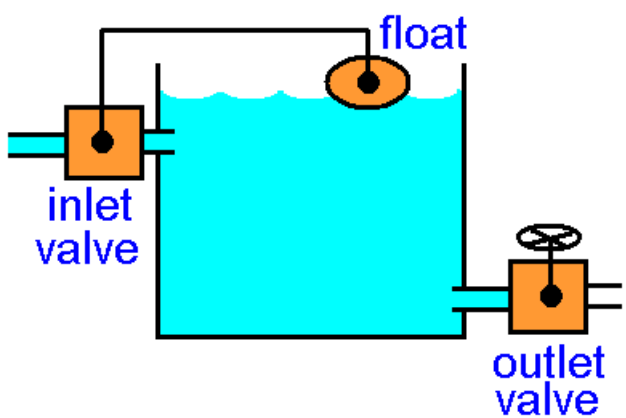
If we measure how fast a machine is moving, we'll know when it's about to overcorrect and we can send a counterbalancing signal, reducing or eliminating overshoot. Or, if we use an overdamped, low-gain servo to quickly get the machine close to its target, and then use a high gain loop that looks back over a longer period and operates slowly to close the remaining gap, we will again get high precision while avoiding overshoot. In other words, if we combine forward-looking, backward-looking and in-the-moment characteristics, we can do better than with a proportional loop alone.

PID stands for **proportional, integral, differential** (or **derivative**). It is a servo design that combines all three response characteristics:

- The output of a proportional loop is real-time;
- The output of a differentiating loop is anticipatory
- The output of an integrating loop is cumulative

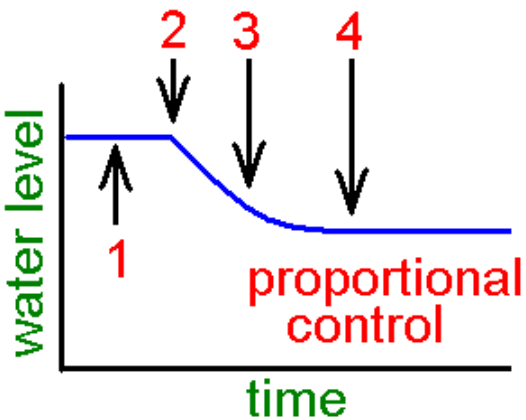
To get an idea how PID works, let's imagine a water tank with a valve at the top to let water in and a valve at the bottom to let water out. The output valve is opened manually or by an electrical signal; the input valve is opened when a float device on the water surface signals that the water level has dropped – servo loop, right. How much the valve is open is *proportional* to the position of the float.

Let's assume that the output valve is always either closed or fully open, so we're dealing with an on-off situation there. When the tank was first filled, the inlet valve was open until the float rose high enough to turn it off.



This established a “normal” water level as shown at **1** below.

At **2** we see the moment when the outlet valve is opened. Water starts to flow out of the tank and the level begins to drop.

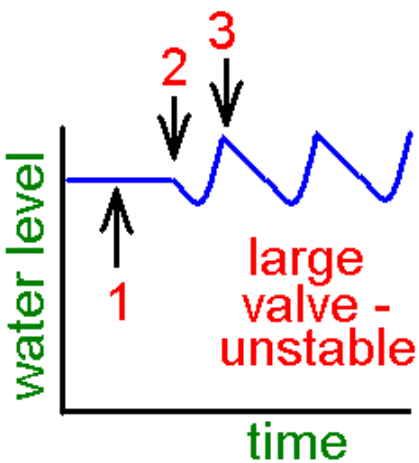


As the level drops, the float drops, opening the inlet valve (**3**). At some point the inlet valve is open far enough that the rate of incoming water equals the rate of outgoing water. From then on, the water level does not change (**4**). However, it does not return to its normal level. This is a very important point: the inlet valve does not completely correct the error between the water's present level and its normal level. The level has to remain low because it is that discrepancy that keeps the float low and the inlet valve open.

This is a basic fact of life with proportional error correction systems: they can't correct completely, because if there were no error there would be no correction.

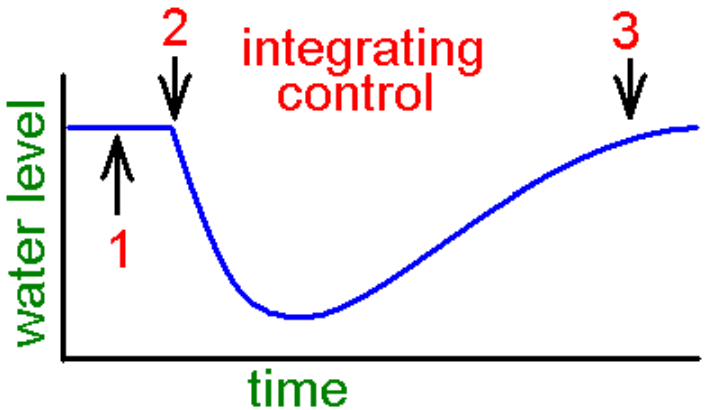
Let's imagine a situation where this error is unacceptable. Suppose it is critically important that the rate of flow at the outlet does not vary. Flow rate depends on water pressure, which depends on water level. So now we must maintain the “normal” water level at all times.

The obvious thing to try is a bigger inlet valve, so just a small drop of the float results in a big gush of water. Problem is, it gushes all at once and overcorrects, as shown at **3** below, resulting in a higher than normal water level.



The problem doesn't stop there. The higher level will slam the inlet valve shut, but the outlet will keep draining water off. At some point the inlet valve will open again, the whole process will repeat, and keep repeating.

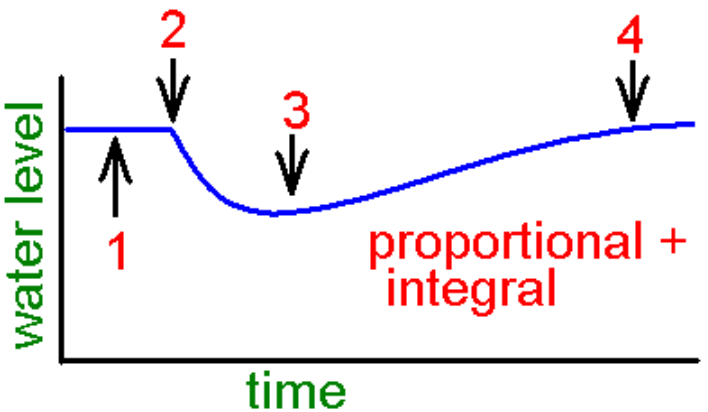
A system where a small error (a slight drop in the float) results in a large correction (a huge inrush of water) would be an example of a “high gain” loop. This shows why, when there's delay involved, you can't necessarily make error as small as you choose by increasing corrective (negative) feedback.



Let's try something else. Suppose we keep the large valve, but put a lot of thick grease in the pivot so it can't open fast. This means its initial correction will be poor, as shown above between **2** and **3**. But eventually it will open enough to pretty much completely return the water level to normal. Since it's slow, we can make it as large as we like. It will just keep opening more and more, however long that takes, until the water level is back where it belongs. By virtue of its slow response, it's stable. Its can be high gain, so it will correct almost 100% of the error – eventually.

A valve whose position is a result not only of the *amount* of error but *how long* the error has existed, a valve whose position represents the accumulated error over time, is an *integrating* valve.

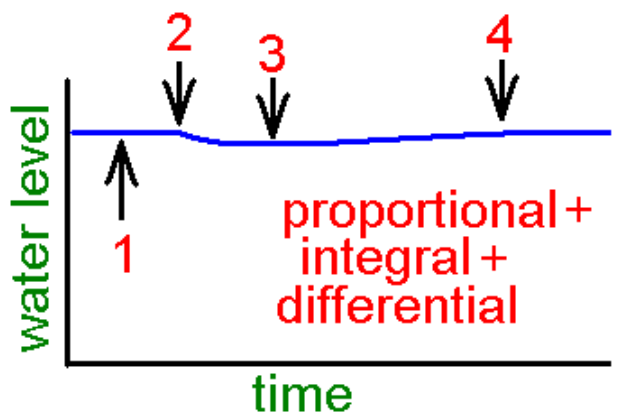
What can we do about that large error at the start? If we have both kinds of valves, proportional and integrating, and make the float operate them both, the system will behave as in the illustration below. We'll get the rapid partial correction of the fast, low-gain proportional valve (**3**) plus, eventually, the almost complete correction of the slow, high-gain integrating valve (**4**).



Of course, when the integrating valve finally opens and the water returns to its normal level, that will close the proportional valve, so only the integrating valve is active.

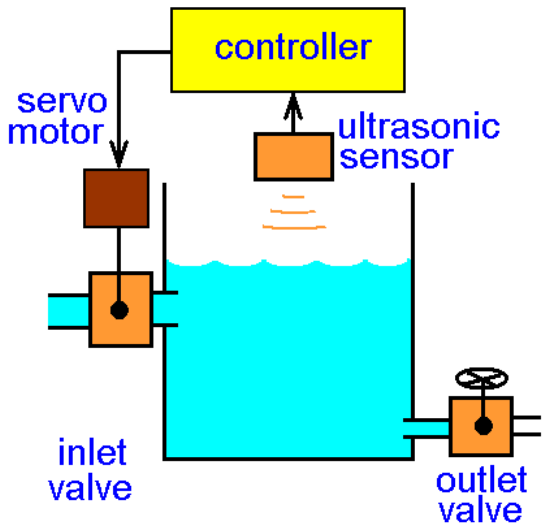
We keep having a stubborn problem with that initial level drop. What we need is a small valve that looks for a level that's *changing*. It doesn't even have to care what the actual level is, because we've already got proportional and integral valves to take care of that. But whenever it sees the level beginning to drop, it should immediately open all the way. And when it sees the level beginning to rise, it should immediately close so it doesn't fight what the other valves are trying to do. In other words, it should not respond to where the water level is, but where it's headed.

With a differential valve quickly acting to counteract any sudden change in water level, we can safely increase the size of the proportional valve to increase loop gain, and take some of the grease out of the integrating valve to speed it up. In a sense, the differential valve doesn't act as a corrective on the water level so much as a corrective for the other two.



To summarize: in the illustration below left, **1** represents the normal water level when the lower valve is closed. At the moment when the lower valve opens, **2**, the level starts to drop. The differentiating valve immediately opens, then closes as the proportional valve takes over (**3**). Meanwhile, the integrating valve slowly opens (**4**), re-establishing the normal water level. We have finally achieved an almost perfectly stable water level.

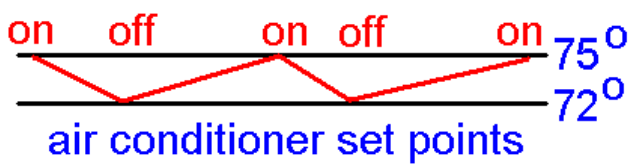
In a real system, we won't be using three mechanical valves. We'll use a single large valve operated by a fast servomotor. The water level sensor will likely be a device that bounces ultrasonic audio waves off the water to gauge its level, like a bat's sonar. The controller will take that single digitized output and, using DSP, mathematically derive separate proportional, integral and differential numbers.



CNC machine control involves the same considerations. Machines are massive, so when they move, they overshoot. A high loop gain proportional control is unstable. A stable integrating loop is slow. But when both are combined with a differentiating loop, the result can be, say, a 100-lb milling turret whipping around as if it weighed five ounces. No human operator could match it for speed and precision.

SET POINTS AND HYSTERESIS

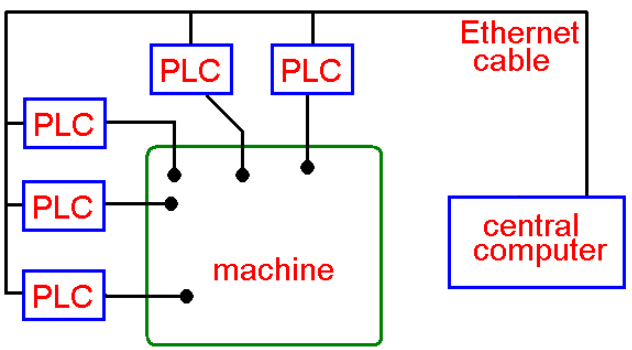
Now that we know how to achieve precise control, let's talk about situations where we don't want it. For example, if an air conditioner tried to keep a room at a precise temperature, it would be turning on and of several times a minute. Most people are perfectly comfortable if the temperature is kept within a range of a few degrees. An air conditioner that turns on when the room reaches, say, 75^{o} and off when it reaches 72^{o}, cycling perhaps every 7-8 minutes, works much more efficiently and lasts much longer. For a general-purpose water supply, a pump that turns on when a tank is almost empty and off when it's almost full is much more practical than one that turns on for a fraction of a second every time somebody draws a teaspoon of water.



In this kind of situation, upper and lower **set points** are established, one to turn something on and the other to turn it off. You'll recognize this as our old friend **hysteresis**. This can be achieved electronically through the use of two comparators and an SR latch, or an analog temperature sensor and virtual set points in software.

DISTRIBUTED CONTROL

As machine control devices have become smaller, cheaper and more capable, it has become common to break the central controller for a factory or large machine into little PLCs placed close to the machine parts they control, and to link them all to a central computer via an Ethernet cable. This allows a human operator to keep tabs on all parts of the system, and to download program changes to the distributed PLCs whenever required.



WIRELESS CONTROL

The newest thing is wireless control, where the Ethernet cable is replaced by radio signals. Factories and other large users have been slow to adopt wireless control because of concerns over reliability. When you're wirelessly controlling a toy helicopter, it doesn't matter much if the radio link is lost and the copter crashes. When you're controlling a million-dollar machine that could destroy itself and the building it's housed in, it matters a whole lot more.

But, then, wired control isn't foolproof either. A connection can break, an Ethernet plug can come loose, a forklift can snag a cable and rip it out. Wireless controllers have improved to the point where their reliability is on a par with wired. Moreover, with PLCs now located at the machines they control, there's enough local smarts to handle a situation where the RF link temporarily goes down.

Wireless control solves a lot of problems. The first is money. In a large factory, the cost of interconnecting the control systems often exceeds the cost of the PLCs themselves. This is especially true if wiring must be run in conduit, which some local codes require even for signal wiring.

The second is ground loops – they don't exist. That's a major plus. In a highly integrated system like a widget factory, information has to be shared among widely separated machines. For example, if the widget packaging machine comes to a screeching halt for some reason, the widget making machines at the other end of the factory have to know about it, so they can stop making widgets until the packaging problem is fixed. A fully wired factory is a groundloop nightmare, which wireless control eliminates.

The third is flexibility. A wired system is like streetcars in a city; if we want to change a route, we have to lay new track. A wireless system is like a buses; if we want to change a route, all we have to do is tell the bus driver to take a different street.

The fourth problem solved is power supply and hot-chassis situations. Each wireless controller is powered locally, at the machine it controls, where there's lots of power available anyway. At the central computer, signals in and out are naturally low-voltage. If a hot-chassis problem arises somewhere in the system, it will involve only a local PLC, and will be easy to track down.

In systems with distributed PLCs, the wired or wireless links aren't doing real-time control anyway. Most of the time, they're gathering data and sending it “home” for archiving and analysis, or giving alerts (e.g. “the packager is running low on glue”) that can afford to wait for the data to be re-sent if it doesn't get through the first time.

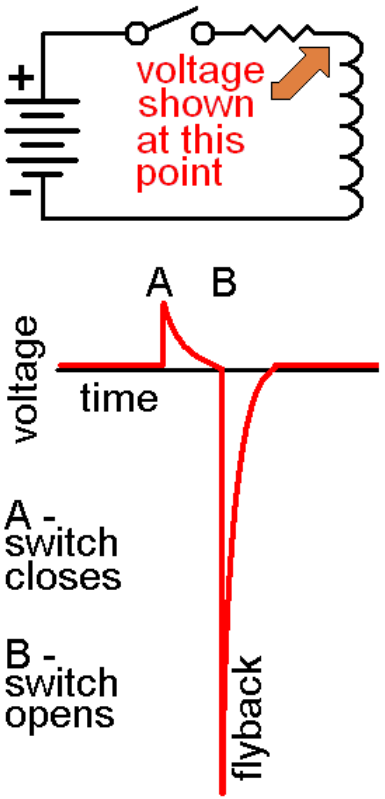
FLYBACK SUPPRESSION

I'd like to close the machine control discussion with a few paragraphs about flyback suppression. We covered this in EM2, which was a while back; a refresher is in order. In machine control applications, you can hardly turn around without running into a flyback issue.

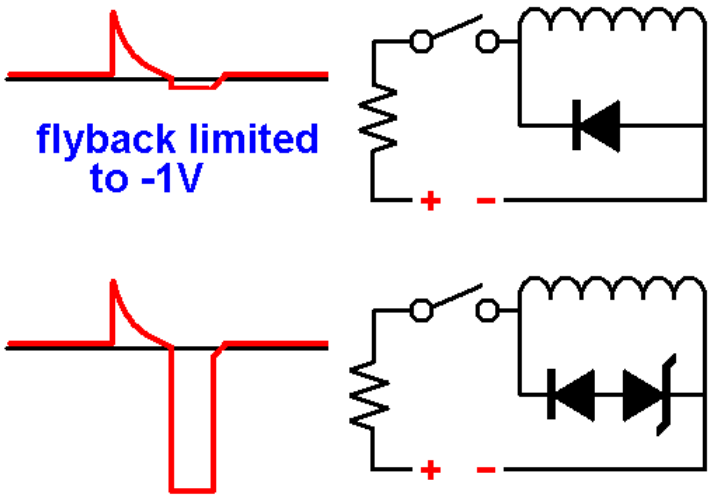
In an inductor, voltage across the inductor is proportional to the *rate of change* of current through it. Steady current in any amount will not cause a voltage to be developed (or would not, if we were dealing with an ideal, zero-resistance inductor).

When a current-carrying inductor is suddenly disconnected – when the circuit is suddenly opened – the current must fall to zero in an instant. The only way this can happen is if an instantaneous, nearly infinite reverse voltage is developed across the inductor's terminals. And, indeed, this is what happens. The collapsing magnetic field in the inductor itself manufactures that voltage. It's called a **flyback**. The illustration at right shows the idea, but it would be difficult to accurately graph an actual flyback – it's often thousands of volts.

In machine control, inductive circuits are being broken all the time. Though you're not often working with inductors as such, every motor, every relay, every transformer, every long wire run has inductance. The constant sparking of contacts can wear them away or, worse, fuse them together. Even if the machine is built to handle the flybacks, the fact that they radiate intense electromagnetic pulses is cause for concern. So a tech has to know how to suppress flybacks.



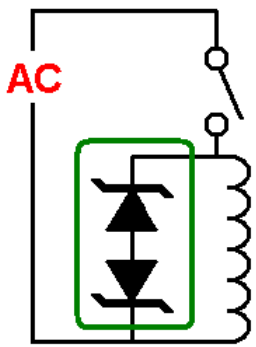
Throughout these EM study guides, I've been showing flyback suppression diodes as in the first illustration on the next page. This is the standard and most effective way to suppress flybacks in DC circuits. When the inductor is energized, the diode is reverse biased, so no current flows through it. When the switch is opened, the coil starts to generate a voltage in the opposite direction. The diode conducts, snubbing the flyback. (Flyback diodes are often called **snubber** diodes.)

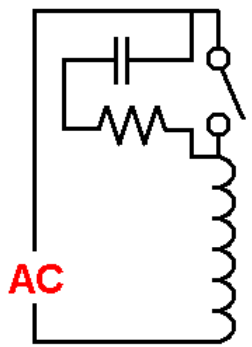


Because the high-voltage flyback is not permitted to develop, the field does not collapse instantly. In some applications, this results in sluggish operation. Adding a zener diode permits a flyback of several volts, allowing the field to collapse quickly.

Zeners especially designed for transient suppression should be used – the P6KE series from Littelfuse, for example. In the circuit above, the zener becomes the snubber. The rectifier keeps current from flowing through the zener when the inductor is energized.

I haven't said much about snubbing flybacks in AC circuits, which will be much more common than DC in machine control applications. A diode cannot be used because the voltage polarity alternates; the power source would be shorted during half of each cycle. In this case an RC circuit or back-to-back zeners must be used to snub the flyback. Flyback suppressors with both zeners in one package are available. Their conduction voltage threshold must be higher than the peak (not RMS) voltage in the circuit, plus a safety margin. In a 117V circuit, 250V would be appropriate.





In the case of an RC snubbing circuit, it often works better to connect across the switch than across the inductor. In a 117V circuit, appropriate values would be a 10Ω, 2 watt resistor and an 0.05μfd, 600V capacitor. Though it's common practice to connect a capacitor directly across a switch, it's not wise. If the switch closes at a point during the cycle where the capacitor is charged to its maximum (almost 180V in a 120VAC circuit), a very high current will exist for a very short time, pitting switch contacts. The 10Ω resistor is there to prevent that.

End of Chapter 8

EM4 Chapter 9
DISPLAYS AND PRINTERS

Objectives of this chapter:

- *explain why virtually all images are 2-dimensional*
- *describe how a 2-dimensional image is built up from a 1-dimensional, serial data stream*
- *define "raster"*
- *discuss time division multiplexing*
- *list display types and describe their technology*
- *define and list primary colors, describe how other colors are made*
- *describe how various types of color displays are made*
- *describe essential difference among NTSC, S-video, component video, give pinout for VGA connector*
- *list essential characteristics of DVI and HDMI*
- *explain fluorescence and describe how it is used in LCD and plasma displays*
- *discuss LCD and plasma technology*
- *define "color wheel" and describe its operation*
- *discuss electroluminescence*
- *discuss electronic paper*
- *describe how various types of printers work*

As any author will tell you, writing is harder than most people would think. The hardest part is organizing the material – deciding what to group with what, and which concepts have to be presented before other concepts so it all makes sense.

My original intention was to include video monitors with computers in chapter 6, but I realized that there are lots of kinds of displays, and a monitor has less in common with a computer than with a meter's segmented number display, an illuminated road construction sign, a watch's LCD display, a smartphone screen and so on. So I've lumped all digital display devices, including printers, into a chapter of their own.

Why are monitors and printers in EM4, the digital study guide? Because these days, analog TV is only hanging on in cable and closed-circuit systems. Everywhere else, it's all digital. And printers have always been digital.

Beyond simple on-off LED indicators, most images have three things in common:

- they are flat 2-dimensional images
- to send them from one place to another, they must be converted to 1-dimensional, serial data streams
- they usually involve time-division multiplexing

2-DIMENSIONAL DISPLAYS

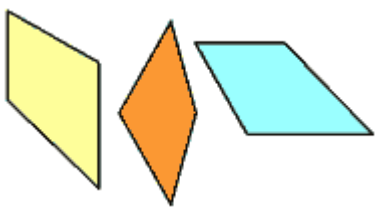
At the risk of belaboring the obvious, nearly all electronically generated images are 2-dimensional. They exist in a flat plane, with no depth. With current technology, there is no such thing as a true 3D image outside the laboratory. You have never seen a 3D image on any video or movie screen.

You: Nonsense, I see 3D video images all the time.

Me: No you don't.

Oh yes I do.

Okay, tell me what the image to the right is.

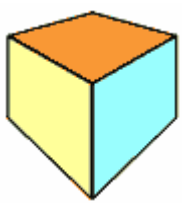


Three trapezoids.

2-dimensional trapezoids, with no depth?

Right.

Okay, now tell me what the image at right is.



It's a cube.

No it's not. It's three trapezoids.

No, it's a cube. Look at it, it has depth.

It does not. It's the same three trapezoid images as before, rearranged to look like a cube, and displayed on the flat surface of your screen, with no actual depth at all. They trick you into thinking there's depth because you've seen so many real cubes in your life. This tendency is so strong that art students have to be trained to turn off their interpretive vision and look at shapes as they really are.

You're probably reading this on an LCD display. The image part is a flat, thin, 2-dimensional array of microscopic light valves. Everything you see is in the same plane.

Computer-produced video images, and for that matter, photographs, printouts, portraits and prehistoric cave paintings are essentially X-Y graphs. They are drawn on a flat surface. When a computer creates, say, game graphics, it uses the rules that artists and drafters spent centuries discovering to distort 2-dimensional shapes so they look like they're slanted at an angle to the viewer.

It seems reasonable to predict that electronic displays will continue to be 2D for a long time. A real, actual 3D image is orders of magnitude harder to produce – ask a sculptor – and there's seldom any point, because a 3D image winds up as 2D images on your retinas anyway.

3D **stereoscopic** images, with or without glasses, are no different; the sense of depth is an illusion. When you see the Grand Canyon in a theater, it looks like it's miles deep – but you know perfectly well that you're actually looking at a flat (or slightly curved) movie screen.

We're so used to 3D scenes represented by 2D images that we hardly ever think about what we're really seeing. With the growth of the 3D stereo (often abbreviated **3DS**) industry, I thought it was worth stressing the point that, with very few exceptions, displays are and probably will continue to be 2D.

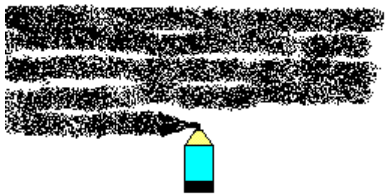
That means the technology boils down to three issues:

- How do we make X-Y images electronically?
- How do we store them in memory and send them from one place to another?
- How do we get them to be in color?

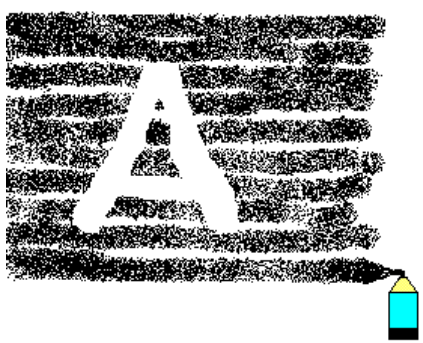
We'll look at the second issue first, just cuz this is my study guide and I will darn well do as I please. I can be such a jerk.

THE 1-DIMENSIONAL DATA STREAM

An analog signal or digital data stream is not 2D. It's a linear, 1-dimensional thing. If you want to make a picture, you have to build it up line by line.

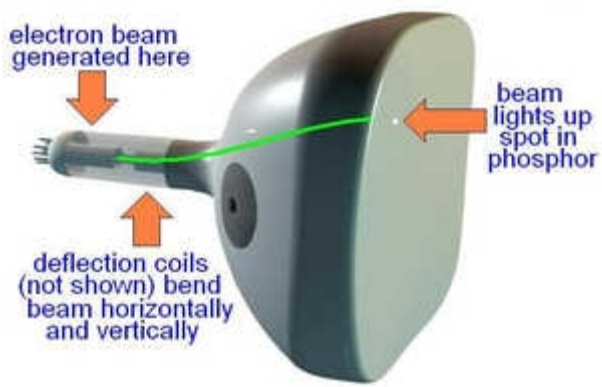


In electronics, creating a 2-dimensional image is a lot like painting a wall using a spray paint can. You start at the upper left, push down the button and move to the right. When you get to the end of the wall, you' made a painted stripe along the top. You take your finger off the button, jump back to the left just below the first stripe, push the button and move across to the right, painting a second stripe. You keep on doing this until you get to the lower right corner. At that point you've covered the whole 2-dimensional wall, but to do it, you moved the spray can in straight, 1-dimensional lines.

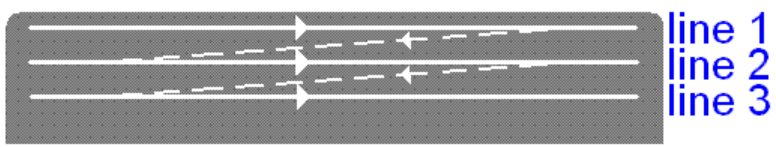


Now suppose, as you moved across the wall, you had occasionally let up on the button, leaving unpainted places on the wall. Suppose you had done it in such a way that the unpainted areas formed a recognizable pattern. This is how a 1-dimensional data stream paints 2-dimensional pictures.

The display of a traditional analog television consists of a **cathode ray tube** or **CRT**, a big glass bottle with no air inside, an electron beam emitter at the back and a nearly flat face at the front, coated on the inside with a phosphor.


electron beam generated here
beam lights up spot in phosphor
deflection coils (not shown) bend beam horizontally and vertically

The electron beam strikes the phosphor, starting at the upper left corner of the CRT face and scanning a line across the screen to upper right. Then it returns to the left, just below the first line, and scans a second line. When it gets to the bottom right corner of the screen, every part of the phosphor has been hit and is lit up.



The entire X-Y scanned image is called a **raster**, which comes from the Latin word for rake, which, you know, makes parallel lines in the ground. I guess the guys who developed this were high-level academics who liked to show off their Latin whenever they got the chance.

Each point on the screen lights up for a longer period than just the moment when the electron beam is hitting it. Once a phosphor is lit up, it continues to glow for a short time. In a CRT, the manufacturer chooses a phosphor that glows for about as long as it takes for the electron beam to come around again during the next scan. This makes the display much brighter than it would be otherwise and greatly reduces visible flicker.

Just as the spray can make an image by turning the spray on and off, a CRT makes an image by turning the electron beam on to make white, reducing it to make gray, and turning it off to make black as it scans across the screen. Occasionally you'll see this referred to as **Z-axis** control, especially in oscilloscopes. 3-dimensional space has three axes, X (horizontal), Y (vertical) and Z (depth). With a 2D image a depth axis is meaningless, but since the electron beam comes toward the viewer from the back of the tube, Z-axis has come to mean electron intensity control, specifically the point-by-point brightness modulation of an oscilloscope trace or video picture.

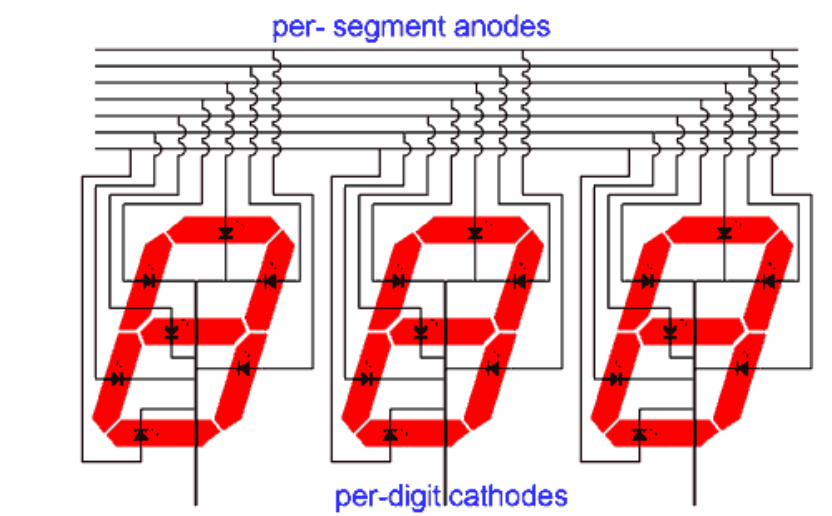
We'll return to this discussion later, because there are a lot more things to say about video – color, for instance. The main thing I wanted to explain at this point is how a 2-dimensional image is converted to a 1-dimensional signal, and reconstructed into a 2-dimensional image.

MULTIPLEXING

Post-CRT displays consist of discrete segments. Signals are sent to all the segments, but not at the same time. For example, consider the arrangement of 7-segment digits in the illustration at right. If this were a clock, there would be six digits. In the left digit there would be only two active segments, because the only number that digit will ever have to display is 1. In all the others, all seven segments are used. In addition, there are colons between the hours and minutes and between the minutes and seconds. In all, there are 39 segments. That would suggest 39 wires between the clock circuitry and the display, plus a common wire.

Instead, one side of all the segments in a digit are connected to a common wire, so there are six common wires, one per digit. The other side of all the similar segments in all digits – all the center segments, for example – are wired to a single wire, so there are seven segment wires. 7 segments + 6 commons = only 13 wires, plus a hot and a common wire for the colons which are always lit.

You know that movies are a series of still pictures, projected in rapid sequence, and your eyes interpret this as continuous motion. In the clock display, it appears that the segments are continuously lit, but they're not. The segment and common wires are scanned, and each digit is lit for 1/6 of the time. Scanning repeats at such a high frequency that your eyes see only continuous illumination.



I've mentioned multiplexing before, but it's time to be more specific, because there are a number of kinds of multiplexing. The kind referred to here is **time-division multiplexing**. This is also used by telephone companies to fit lots of conversations into one communication channel.

The more complicated a display, the more important multiplexing becomes. Without multiplexing, a hi-def video display would require several million wires! Instead, every little dot has a separate transistor driver circuit, right in the screen, at the dot, and data to the millions of transistors is scanned in via time division multiplexing.

Now that we've set the stage, we can talk about the various ways electronic images are created.

DISPLAY TYPES

The various display types represent a range of answers to the basic question, “How do we get this thing to light up?” Some displays create lighted patterns:

- incandescent and LED arrays
- CRT monitors
- electroluminescent panels
- plasma displays
- organic led panels

Others start with a lighted area, then selectively darken or block the light to create images:

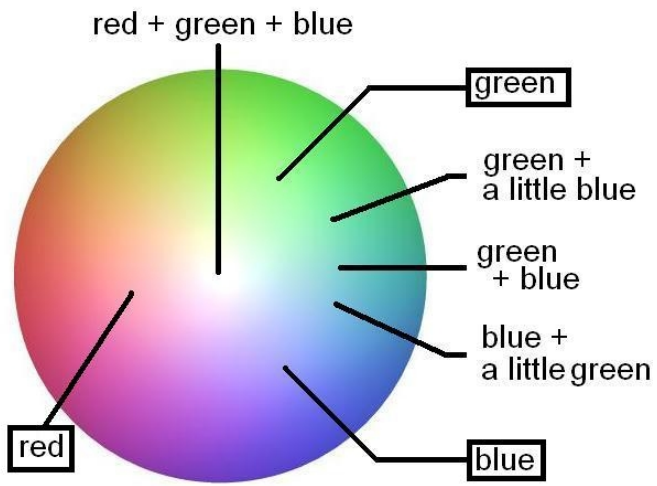
- LCD and LED-illuminated LCD displays and projectors
- mechanical road signs
- printers

VIDEO

I'm going to switch things around and start with the most recent and most complicated complicated display technology first, video screens. As these displays get cheaper, they're replacing simpler displays in many applications. Photocopiers have used them for years, and they will increasingly show up in other appliances that have traditionally had nothing to do with video.

COLOR

Pretty much any shade of color can be produced by mixing light of the **primary colors** red, green and blue. For example, if you mix blue and green, but not red, you get aqua. Mixing all three gives you white. A color video image is simply three video images in the primary colors, superimposed to make a single color image.

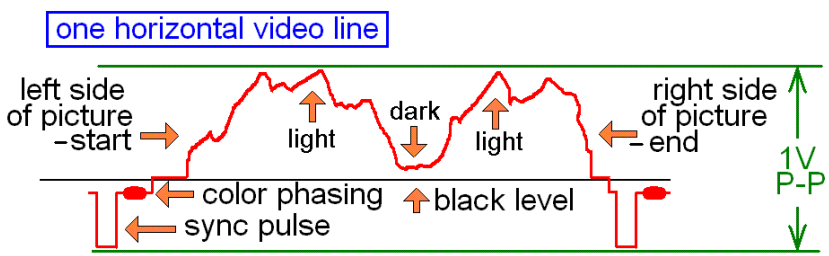


VIDEO COMMUNICATION STANDARDS

I've already described how a CRT works. CRTs are mostly obsolete. They are only important because certain standards and methods from the CRT days are still with us.

The standard broadcast system, established in the 1940s, is based on a picture made up of 525 horizontal lines, with a new picture sent 30 times per second. Each picture is a **frame,** just as in film, thus the **frame rate** was 30Hz. When color was added in the 1950s, the frame rate was lowered by a small fraction of a Hz.

A **composite** analog video signal, one with all the different aspects combined into a single varying voltage, is a waveform with about 1-V peak to peak amplitude. Somewhere near the middle of that volt is a level that corresponds to black. Higher voltage points correspond to lighter grays, until the signal reaches its highest level which corresponds to peak white. Going in the other direction, voltages below the black level are used for pulses that synchronize the system.



There are three widely used analog standards for sending video signals to a monitor, plus several analog ones. Two of the analog standards, NTSC and S-video, are so limited that I'm only going to mention them in passing so you'll recognize them when you run into them.

NTSC

When you see an RCA jack with a yellow insert on a monitor, that's an NTSC connector. NTSC is composite video as it has existed since the 1940s. It is carried on a single coax cable and all parts of the system have a matched 75-ohm impedance. (Most other radio-frequency systems operate with a 50-ohm impedance.)

In an analog display, resolution is measured in **lines**. Of the 525 horizontal lines in NTSC, fewer than 500 actually get on the screen, so the vertical resolution is automatically around 490 lines. Horizontal resolution depends on the high frequency response of the system; that is, how many brightness changes can be transmitted in the course of a single horizontal line.

Very good NTSC can achieve around 500 lines of horizontal resolution; 350 is more typical. “Lines” means alternating light and dark lines, so 350 lines means about 175 light lines and 175 dark ones.

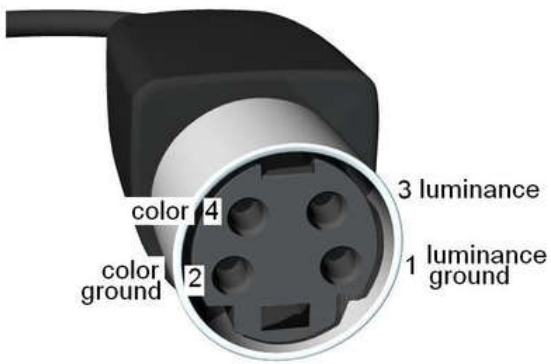
In NTSC, the three primary colors are added and subtracted and manipulated to get three new signals. One is brightness or **luminance**, often represented by the letter **Y**, the others represent color or **chrominance**, represented by **Cb** and **Cr**. The three signals are then recombined to form a single signal called **composite video**.

There were two reasons for all this complication. One, color TV had to be compatible with black & white TV, so the signal had to be based on a luminance signal. Two, it had to fit into the existing TV channels, which means it had to make efficient use of the bandwidth, at a time when digital compression techniques were not available. The eye can see much finer brightness details than color details. This meant the chrominance signal could be bandwidth-limited, saving spectrum space, without making the picture look blurred.

Given the technology of the 1940s, NTSC color video was a brilliant achievement. But it has lots of limitations, one of which is that an NTSC display hasn't got anywhere near the resolution you'd want for text. Also, combining all the various aspects of the video into a single signal caused problems that TV production people have to deal with to this day. For example, any image with strong horizontal lines can cause trouble. If they're shooting a window with blinds, it may have to be shot at an angle so the lines won't be horizontal. A plaid shirt or jacket can look like a hippie-era drug trip.

As long as “video” meant “broadcast,” we were stuck with NTSC. The standard was like a straitjacket for more than 30 years. But when the image is made by a computer, and the monitor sits right next to it, we're perfectly free to adopt a different standard, limited only by the state of technology. Once computer video was separated from broadcast video, the race was on to improve picture quality. That race continues today.

S-VIDEO



The first step away from NTSC was to partially “unbundle” the signal by splitting luminance and chrominance information apart and sending them separately over two coax cables. This is the **S-video** standard and it usually involves a mini-DIN connector. Note the separate grounds for each signal, something that's common in high-definition video standards.

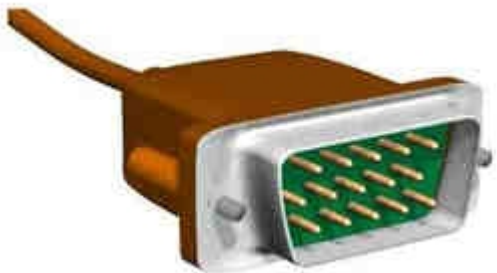
LINES VS. PIXELS

With digitally generated video, the display is literally an X-Y grid of so many squares horizontally by so many squares vertically. Inside the computer, there's a special section of memory reserved for the display, and every square in the display corresponds directly to a memory word or several words that hold the red, green and blue brightness values for that square.

So instead of talking about lines, we now talk about **picture elements** or **pixels**. Each of those little squares is a pixel. In an analog system, the intersection of one horizontal line and one vertical line would be equivalent to a pixel in height and width. Digital video has retained the analog scanning concept: a serial data string begins with a word representing the upper left pixel, then the next pixel to the right, etc.

VGA

VGA, or **video graphics array**, is the analog video standard specifically designed for computers. VGA recognizes that in a self-contained computer system, where we can design the computer-to-monitor interface any way we please, we can keep the various signals separate through the whole system. VGA sends five separate video signals over five individually shielded wires for red, green, blue, horizontal synchronization and vertical synchronization, as well as several control signals. It uses a D-connector with 15 pins, miniaturized so the 15 pins fit in a standard DB9 shell.



With VGA, displays broke away from the old NTSC-imposed resolution limits. The original VGA, introduced with the IBM PS/2 computer in 1987, offered 640 horizontal by 480 vertical pixel resolution, which is a bit better than the best broadcast NTSC and almost double the resolution of the average TV set. From there, resolution has continued to increase:

VGA	**640 X 480**
SVGA	**800 X 600**
XGA	**1024 X 768**
SXGA	**1280 X 1050**
SXGA+	**1400 X 1050**
UXGA	**1600 X 1200**
QXGA	**2048 X 1536**
QSXGA	**2560 X 2048**

...and the beat goes on.

The formats above are in the traditional video **aspect ratio** of 4:5 or close to it, in which the image proportions are 5 units wide by 4 units high. There is also a wide-screen version of each, identified by the letter W, e.g. WXGA, with a typical aspect ratio of 9:5.

Though modern monitors have advanced to higher-resolution formats, the connector is still generically known as a "VGA connector." The pinout is:

1	**Red video**
2	**Green video**
3	**Blue video**
4	**(reserved)**
5	**Horizontal sync shield/ground**
6	**Red shield/ground**
7	**Green shield/ground**
8	**Blue shield/ground**
9	**+5VDC**
10	**Vertical sync shield/ground**
11	**(reserved)**
12	**I^2C data**
13	**Horizontal sync**
14	**Vertical sync**
15	**I^2C clock**

I^2C is a communication standard that, in this case, is used to tell a computer about the monitor's model number, resolution and other details, so the user doesn't have to set those things up manually. Note that the signals are sent over unbalanced, separately shielded cables. In practice, separate grounding of various shields is seldom used; most systems just ground pins 5, 6, 7, 8 and 10 together.

The +5V supply is handy because there are some things you want a monitor to do even when it's shut off – like, for example, watch a pushbutton to see when to turn on. The main data path, representing video, is simplex (one direction only), because there's no reason to send video back from the monitor to the computer.

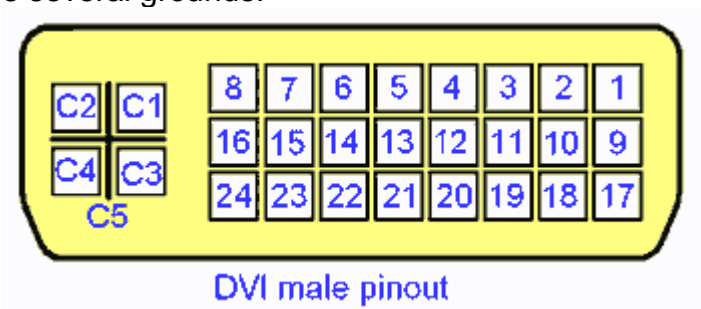
VGA is called **component video** because the three component colors and two sync signals are kept separate, instead of being mixed together as with NTSC.

DVI

In the days of CRT monitors, which are analog, it made sense to change the digital information in the computer to analog before sending it to the monitor. With an LCD or LED monitor it makes no sense at all. The signal is digital in the computer and winds up being digital in the monitor. Converting it to analog to send over a VGA cable only complicates the system and degrades picture quality.

The **DVI** or **digital visual interface** standard was designed to supersede VGA and analog video. The connector is a big, fat, clumsy thing, but it works well. It even has pins for analog video, so if a computer has a DVI connector and you want to run an analog monitor, you can get an inexpensive adapter.

The standard provides for two separate digital links each for red, green and blue, six in all, thus doubling the maximum total data rate. Data is sent over balanced pairs. There are also a number of control and sync wires, +5VDC, and of course several grounds.



DVI male pinout

1 **Data 2, Red 1-**
2 **Data 2, Red 1+**
3 **Data 2 & 4 shield**
4 **Data 4, Green 2-**
5 **Data 4, Green 2+**
6 **DDC clock**
7 **DDC data (I^2C)**
8 **Analog vertical sync**
9 **Data 1, Green 1-**
10 **Data 1, Green 1+**
11 **Data 1 & 3 shield**
12 **Data 3, Blue 2-**
13 **Data 3, Blue 2+**
14 **+5V**
15 **Ground, return for analog sync and 5V**
16 **Hot plug detect**
17 **Data 0, Blue 1-**
18 **Data 0, Blue 1+**
19 **Data 0 & 5 shield**
20 **Data 5, Red 2 -**
21 **Data 5, Red 2+**
22 **Data clock shield**
23 **Data clock +**
24 **Data clock -**
C1 **Analog red**
C2 **Analog green**
C3 **Analog blue**
C4 **Analog horizontal sync**
C5 **Analog ground**

The format treats video as if it were actually being scanned, left to right, top to bottom, like an analog TV, one pixel at a time. Maximum data rate is 165MHz per link, which will handle a 1600 x 1200 pixel display with a 60Hz refresh rate. But since DVI supports two links – two separate batches of Red, Green and Blue data – using both in combination to make one picture will support at least 2048 x 1536 pixels at 60 frames per second. This 60 frame per second (fps) rate is now much more common that 30Hz in non-broadcast systems, a result of people's demand for faster monitors, especially for gaming.

Pin 7 transmits I^2C data from the monitor to the computer, giving the computer information about the monitor's resolution etc. As someone who remembers when all that stuff had to be set up manually (you need to look at a monitor to set it up, and until it's set up, the monitor doesn't work right!) I very much appreciate the auto-configuration feature made possible by I^2C.

DVI does not carry audio. This isn't as shortsighted as it seems. Most new computer monitors don't have speakers and the audio comes from a completely different place in the computer anyway. Plus, if we're talking about analog audio, a good way to screw it up is to bundle it in a cable with digital signals. As far as DVI is concerned, audio exists in a separate universe.

HDMI

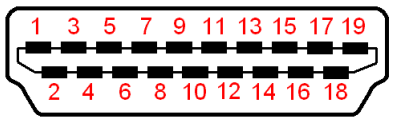
HDMI is a television standard, not a computer standard. However, lots of computers and monitors accommodate it. There are only three video channels, one each for red, green and blue, which are sent over balanced wiring.

Display-wise, HDMI version1 is equivalent to single-link DVI, minus the analog video outputs, plus audio and some control features, with a 165MHz frequency limit. Starting with HDMI version 1.3, a 340MHz data rate is supported, so one link's worth of HDMI is as good as two links' worth of DVI. There's no provision for analog but there is provision for Ethernet and digital audio.

The pinout of a “type A” HDMI connector is:

1	**Data 2+**	**11**	**Data clock shield**
2	**Data 2 shield**	**12**	**Data clock -**
3	**Data 2-**	**13**	**CEC**
4	**Data 1+**	**14**	**HEC data**
5	**Data 1 shield**	**15**	**I^2C clock**
6	**Data 1-**	**16**	**I^2C data**
7	**Data 0+**	**17**	**I^2C ground**
8	**Data 0 shield**	**18**	**+5V, 50mA max**
9	**Data 0-**	**19**	**Hot plug detect**
10	**Data clock +**		

HDMI type A plug



The 340MHz data rate corresponds to a 2,560 x 1,600 pixel picture at 60 fps. If two parallel HDMI channels are used, the total data rate is increased to 680MHz, corresponding to a horizontal resolution of about 4,000 pixels. At those frequencies, the best standard cables only work up to about 50 feet. However, amplified cables and extenders of various types are available, including fiber optic units with a range up to 1,000 feet.

CEC, **consumer electronics control** (pin 13), allows up to ten linked devices (receiver, blu-ray player etc) to be controlled by a TV's wireless remote control. Cool beans.

HEC or **HDMI Ethernet channel** (pin 14), accommodates Ethernet up to 100MHz. A lot of receivers, blu-ray players etc have ethernet functionality, and HEC eliminates the need for separate ethernet cables.

Regarding audio, HDMI's designers decided that this time, they weren't going to adopt a format they would regret; obviously the CD screwup taught them a lesson. So, basic HDMI supports a minimum of 2-channel stereo, but can support up to 8 channels of 24-bit, 196kHz audio, or 1-bit Super Audio, or Dolby Digital, plus other formats. There's also a “return audio” feature where, if you're using a TV with a tuner, its audio is sent back to the stereo system so you can turn it up and really get the feel of Darth Vader's gasping. Audiophiles should be in hog heaven.

HDMI connectors are called **plugs** and **sockets**. The usual reference to male and female would be ambiguous, due to the design of the connectors.

A very well thought-out standard, what could they do to mess it up? Well, how about this: there are *five* connector types. They're all physically different, the pinouts are different, they don't plug into each other.

- Type A: 19 pins, electrically compatible with DVI. Depending on data rate, a DVI to HDMI adapter can be just a pair of connectors with wires between them.
- Type B: 29 pins, with two complete RGB links, electrically compatible with dual-link DVI. A type B link will handle a display with almost 4k of horizontal resolution.
- Type C: a mini-sized type A. The pinout is different.
- Type D: a micro-sized type A, almost exactly the size and shape of a micro-USB port. Meaning that, when the two are side by side on a smartphone, users are likely to break them by jamming a plug into the wrong connector. Nice going, guys.
- Type E: vibration-resistant, for use in cars. Sturdier than type A, with a latching mechanism.

HDMI has a reputation for unreliability. Don't blame the standard, blame the cheap, crummy cables flooding the market. I wouldn't trust a cut-price, no-name import cable beyond about 15 feet. If you install a cable in a wall, and it winds up not working, you've got a real problem on your hands. **Rule of thumb, not just for HDMI but for everything: if you can, plug it all together in the shop, including the actual cables you're going to use, and make sure it works before you take it to the jobsite.**

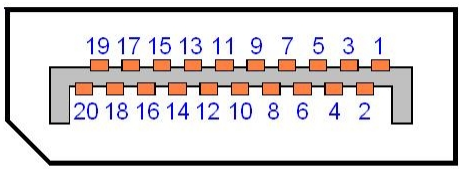
DISPLAYPORT

Like HDMI, Displayport is a television standard. It's hard to guess where technology will take us next, but I think Displayport will become a much-used computer standard because:

- it shares a lot of technology with Thunderbolt, which is a data transfer standard. People have been trying to marry video and computer technology for years, because it's obvious that there's no reason for them to be separate technologies any longer. But there never was a longer or clumsier courtship. Thunderbolt/Displayport may be the high-speed, universal standard needed to pull this off.
- Thunderbolt is already showing up in a lot of computers, especially Apple and Sony.
- Intel is behind it.
- manufacturers can use it without paying royalties.

Displayport is the first "standard" video distribution scheme that recognizes modern video for what it is: digital data. Information is sent in mini-packets similar to what travels around on the web, and there is great flexibility about what gets sent where at which time. Data channels, which like DVI and HDMI are balanced pairs with individual grounds, are called **lanes**, like highway lanes, where the flow of traffic is flexible. No R, G, B wires.

cable end connector



equipment end pinout

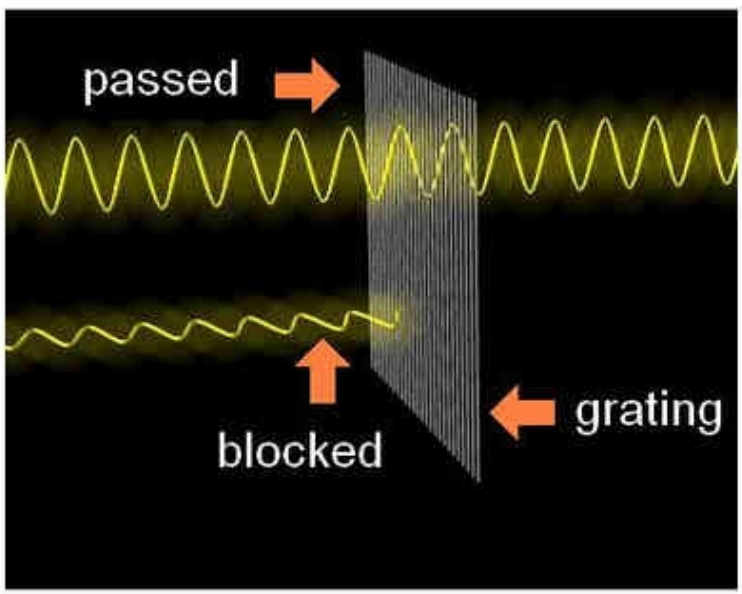
PIN	SIGNAL
1	lane 0 +
2	lane 0 gnd
3	lane 0 -
4	lane 1 +
5	lane 0 gnd
6	lane 1 -
7	lane 2 +
8	lane 2 gnd
9	lane 2 -
10	lane 3 +

PIN	SIGNAL
11	lane 3 gnd
12	lane 3 -
13	config 1 (gnd)
14	config 2 (gnd)
15	aux chan +
16	aux gnd
17	aux chan -
18	hot plug detect
19	power return
20	power 3.3V 0.5A

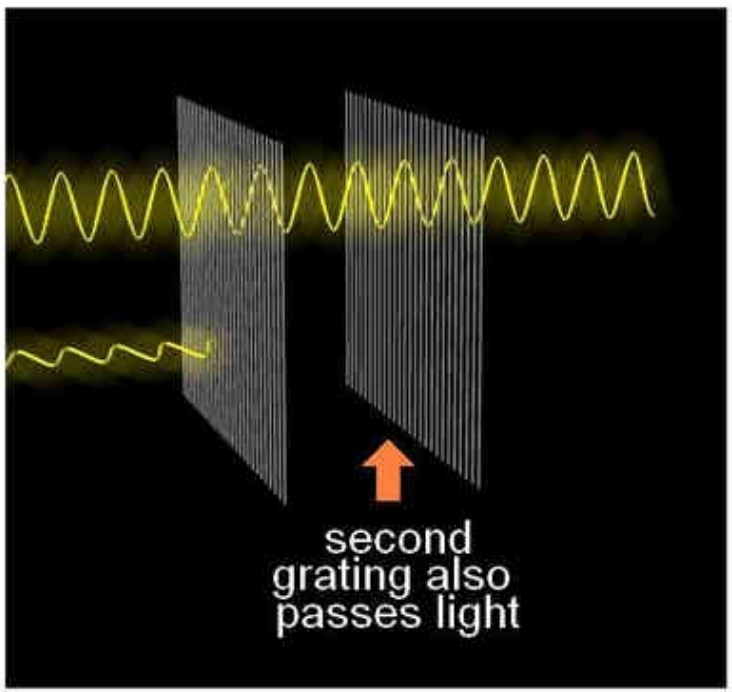
VIDEO DISPLAY TECHNOLOGIES

LCD

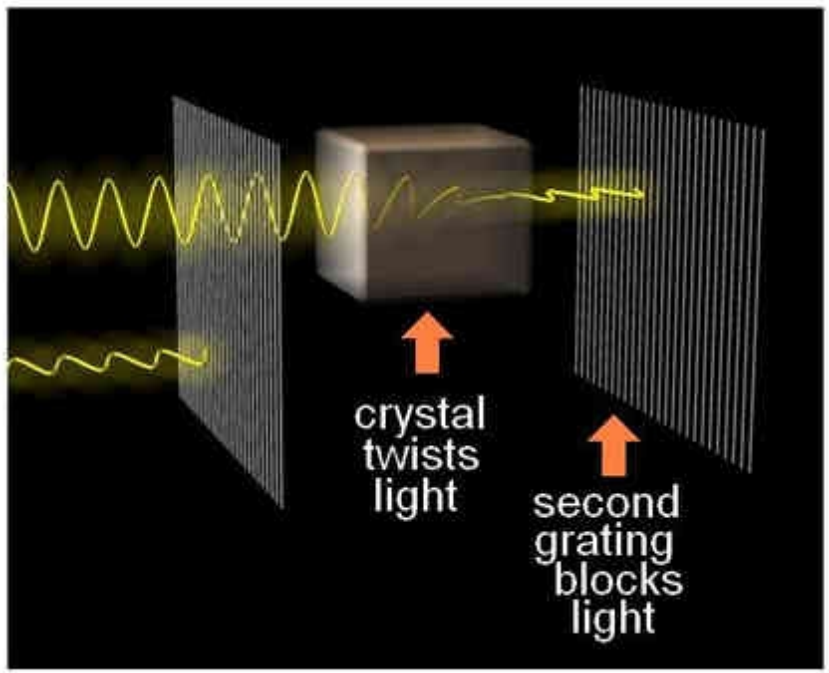
The dominant current technology is **LCD** or **liquid crystal display**, where the 2D display is flooded with light which is then selectively blocked to create an image. An LCD panel makes use of light **polarization**. A polarizing filter consists of a grating of fine parallel lines so close to each other that their spacing is comparable to the wavelengths of visible light. Normal white light waves occur in all orientations – vertical, horizontal and angled. When light is passed through a polarizing filter, it removes the light waves that do not have the same orientation as its own lines. For example, if the filter lines are oriented vertically, only vertically-oriented light waves will get through.



Suppose we place a second polarizing filter in the light path. If its orientation is the same as the first, any light that got through the first filter will also get through the second. But if it is rotated, so its orientation is 90^{O} from the first filter, the light will be blocked.



A liquid crystal material twists light waves as they pass through it. The degree of twist changes if we apply a voltage. If we make a sandwich with a liquid crystal layer between two polarizing filters, we can control whether light that gets through the first filter can also get through the second, by controlling the degree of twist. In other words, we have an electrically operated light switch. If we get a little more sophisticated, and vary the voltage in a linear fashion, we have what we might call a light “potentiometer,” a device that can give us light or dark or any shade of gray.

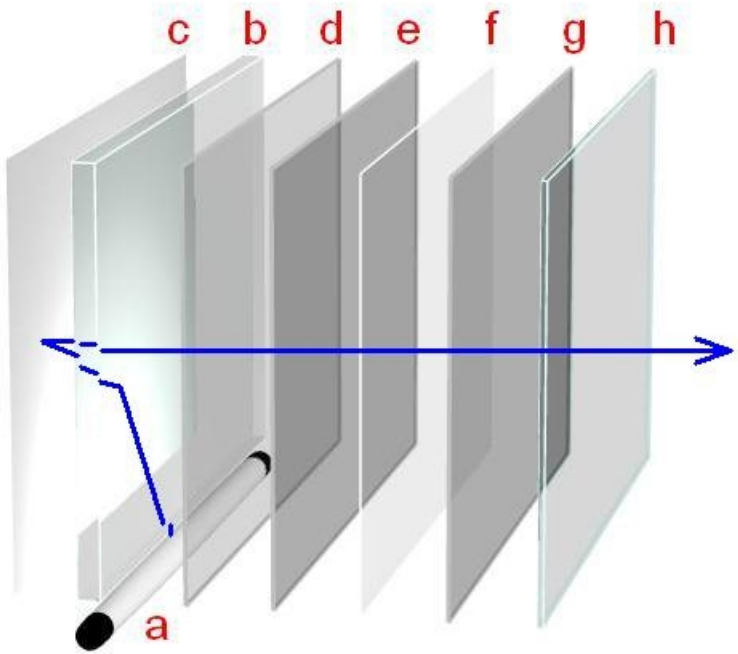


Now let's break the liquid crystal in about a million pieces, arranged in a 1,280 x 768 grid, and supply each piece with independent electrical drive circuitry. We target each circuit individually, sending a signal for that piece alone, using time division multiplexing . This makes a megapixel or about XGA-resolution display.

Finally, let's break each piece into three pieces, and paint red, green and blue filters on them. Now we've got an array of 3,840 subpixels by 768 pixels (or, if we choose, 1,280 by 2,304 subpixels), a color LCD.

FLUORESCENT ILLUMINATION

An LCD display works by *blocking* light, which means we have to put an evenly illuminated white panel behind it. Until recently, the standard illumination source for LCD panels was a fluorescent lamp, which is a tube with a little mercury vapor in it and a phosphor coating on the inside surface. When current is passed through the vapor, it emits ultraviolet light. The UV light strikes the phosphor, bumping some electrons into a higher-energy orbit. (See, I told you in the EM1 study guide we'd come back to that atomic stuff.) These electrons quickly fall back to their regular orbits, emitting light, but at wavelengths in the visible spectrum. If the particular mix of phosphors is chosen correctly, the lamp will emit broad-spectrum white light.



The illustration above is an exploded view of an LCD panel; the blue arrow traces the light path. A fluorescent lamp at the bottom (a) edge-lights a thick plastic panel (b) which distributes the light over a wide area. This light evenly illuminates a white coating on the back of the panel (c). The white coating then reflects and distributes the light. A **lenticular** lens (d) collects the light that would normally go up and down and redirects it forward, where viewers are most likely to be. The light then passes through a polarizing film (e), the liquid crystal layer (f), the other polarizing film (g) and on to the viewer. A sheet of glass (h) protects the display.

"Lenticular," by the way, means a lens that is made up of many long, tiny ridges that form microscopic cylindrical lenses. Unlike regular lenses, they focus light in the horizontal plane, or the vertical plane, but not both.

LED-ILLUMINATED LCD

LCD displays are being replaced by "LED" displays. I put "LED" in quotation marks because these are still LCD displays. The difference is that, in place of the fluorescent tube, we use LEDs. Just a different way of getting our wash of white light. A panel may be edge-illuminated, but the LEDs may also be spread around behind the panel; many phones are lit this way.

LED-illuminated LCD panels have the same advantages LEDs have generally over fluorescent lamps:

- simpler circuitry
- no high voltages
- no hazardous waste
- somewhat better efficiency
- if red, green and blue LEDs are used, no gradually dimming phosphor

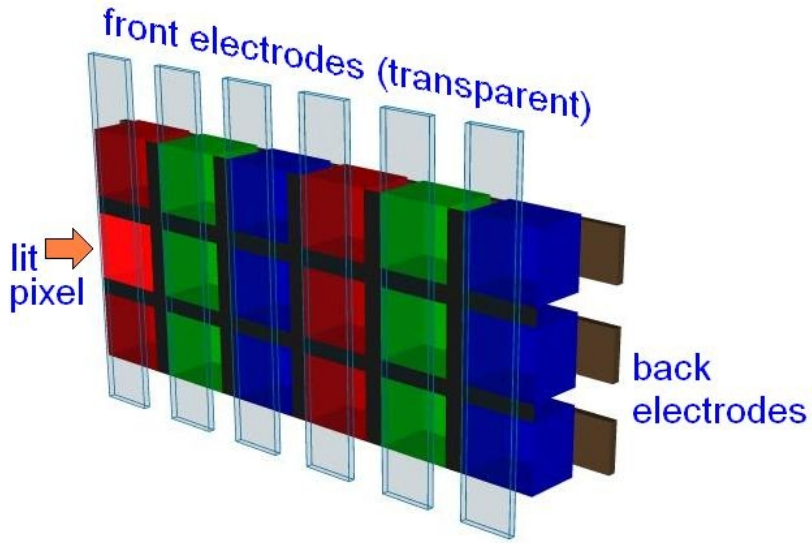
OLED DISPLAYS

These are true LED displays, where the picture is created directly by millions of tiny LED pixels. **OLED** is short for **organic LED**, which doesn't mean there are millions of live bacteria in the display – “organic” just means hydrocarbon-based chemistry.

OLED displays to date are expensive and not very large, but this is an emerging technology, barely out of the laboratory. Since we're not dealing with a thick edge-lit piece of plastic, OLED displays can be nearly as thin as paper and likely can be made flexible. My guess is that OLEDs will begin to seriously compete with LCDs around 2015.

PLASMA

A plasma display works like a fluorescent lamp: current passes through a gas that includes mercury vapor, creating ultraviolet light, which in turn lights up a phosphor. In a plasma display, the mercury vapor chamber is divided into millions of little boxes behind the glass. One side of each box, the side toward the viewer, is open. The other five sides are coated with red, green or blue phosphor. An X-Y matrix of electrodes selectively activate the ultraviolet light. PWM is used to control the brightness in each box.



Plasma displays may or may not be obsolete, depending on who you talk to. Many manufacturers have ceased production of plasma displays to concentrate fully on LCD. Others continue to hang in there, because plasma displays have several advantages over LCDs:

- brighter whites and darker blacks
- fast refresh times, good for fast motion
- wider range of viewing angles than most LCDs
- lower price than equivalent LCD displays

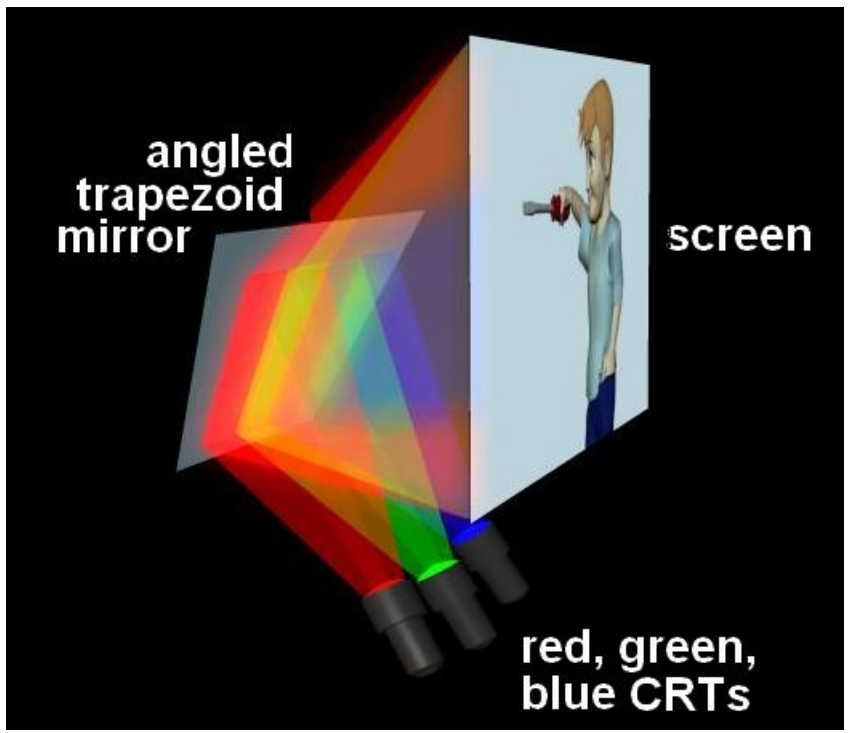
Plasma displays have had two drawbacks: they tend to consume about twice as much power as an equivalent LCD display (though new ones are narrowing the gap) and if a stationary image is displayed for a long time, it will "burn in," meaning you'll still see a ghost of the image when other images are displayed. Plasma displays will probably continue to occupy a modest market niche, at least for a while.

VIDEO PROJECTORS

Video projection can be either **front projection** or **rear projection**. Front projection is like a traditional movie theater: a projector above and behind the viewers projects an image on a white screen. Rear projection, where the projector is behind the screen, requires a translucent screen and a significant amount of space behind the screen. Which is better depends on the situation. The projector is the same in either case; most projectors even have a switch to flip the picture horizontally, so it won't appear backwards in a rear-projection installation. The difference is in the screen.

The problem with video projectors has always been getting enough light onto the screen. The earliest fully developed unit was probably the front-projection system made by Advent, which used three high-power CRT displays, a focusing arrangement, and a specially made screen to eke out enough light for a passable picture in a dark room.

Projection CRTs continued to get better and brighter. The first really successful large screen TVs, shown below, were rear-projection units combining three CRTs, a system of mirrors, and a special plastic screen incorporating a lenticular lens.



Rear-projection TVs needed a depth of up to two feet to accommodate the mirrors, but that's a lot less depth than required by a traditional rear projection system, and no deeper than a much-smaller traditional CRT display.

CRT projection systems are almost obsolete. Nearly all projectors are either LCD or **DLP** machines.

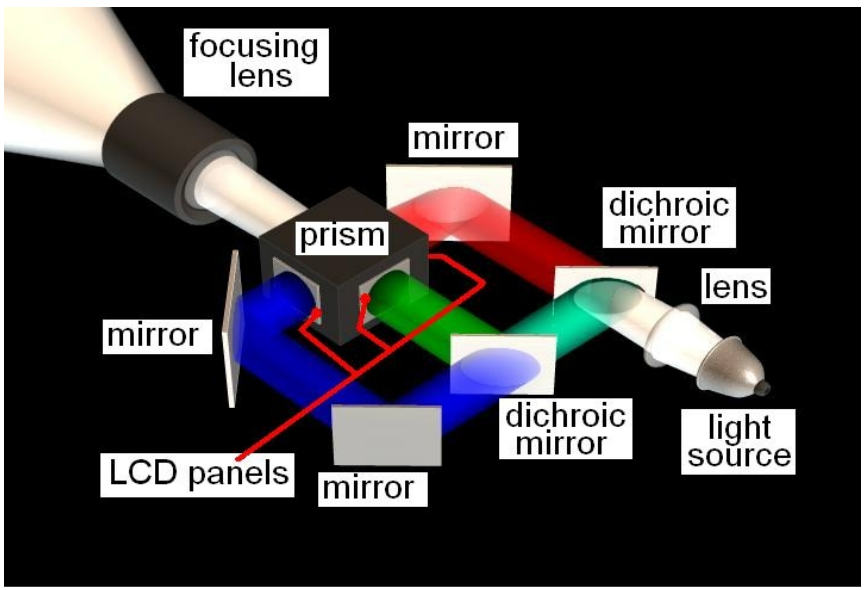
LCD PROJECTION

An LCD projector is essentially a small LCD panel with a very bright light behind and a lens in front. In fact, some people hack LCD displays, removing the illumination panel and placing the LCD on an old overhead projector to get a video projector on the cheap.

In an actual, proper projector, the LCD is quite small, meaning it's hard to make enough pixels for 3-color high definition. Also, color dyes absorb light and turn it into heat – a lot of heat when we're talking about a projector. These problems are solved by abandoning the red, green and blue subpixels and just making a black & white LCD.

To get a color display, deluxe and high-power projectors use three separate panels with red, green and blue filters. This introduces the problem of insuring that the three separate images line up, overlaying each other perfectly. This is called **convergence**, a term borrowed from the days of CRT color TV. Various manufacturers have different mechanical and electronic ways of dealing with the issue.

In 3-LCD projectors, **dichroic mirrors** split white light into red, green and blue components. The three beams are passed through three LCD panels, then a prism recombines them into a full color image. ("Di" means two, "chro" means chroma as in color. A dichroic mirror passes some wavelengths and reflects others, splitting the light spectrum. This technique has long been used to separate white light into its component colors.)



A lot of work has been going into making projectors using LEDs and lasers, and these are beginning to be widely available. In these cases, separate red, green and blue light sources are used, making the light-wasting splitter mirrors unnecessary.

Color Wheels

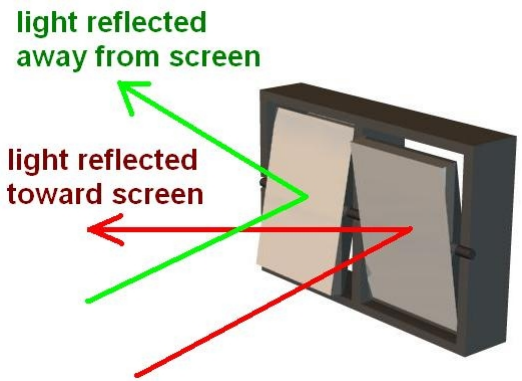
Lower priced projectors use a single LCD and a rotating **color wheel**, with several sectors of different-colored filters. For every video frame, the LCD displays at least three pictures in sequence. The color filters then turn these into superimposed red, green and blue images. I say “at least” because many color wheels have additional sectors. For example, a clear section allows the projector to create really bright whites.

The filters are actually dichroic mirrors; if you look through them you'll see one color and if you look at them you'll see another. By reflecting unwanted colors away instead of absorbing them, the wheel stays cool.

The color wheel system is called the **field sequential** system, and it's a very old idea. Way back in the 1920s, when television was in its infancy (no sound, no synchronization, no regular broadcasts), people were creating experimental color TVs using color wheels.

If you move your eyes while watching a 30- or 60-frame per second image from an older single-LCD projector – as when you're watching something move across the screen – you will see **fringing**, where colored fringes appear around objects. That's because your eyes move between the times when the red, green and blue images are projected. Modern projectors reduce fringing by dividing the wheel into more sectors and/or spinning it faster, so each color is projected many times for each frame. This requires a pretty fast LCD, whose image must change with each sector. 3-LCD projectors, of course, do not have the problem.

DLP PROJECTION



DLP is based on a large array of microscopic mirrors smaller than a human hair, each of which can pivot to direct light in one direction or another. In one position, light hits the screen. In the other, it hits a black absorptive surface.

Brightness is controlled by the duty cycle, i.e. the percentage of time the mirror spends in each position. Add a color wheel or use three DLP arrays, and you get color.

HUGE LCDs

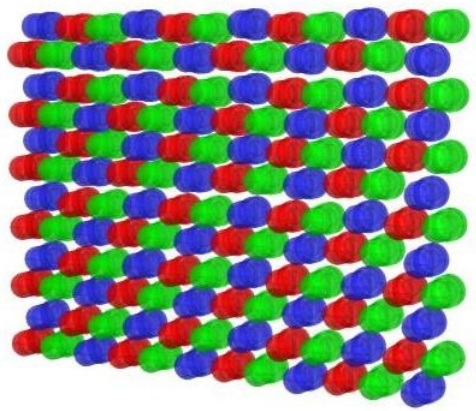
So, you ask, where is the standard, direct-view LCD display in all of this? Why should we need projectors for anything but the largest displays? Because, I answer, large, 8-foot and up LCD displays have only recently become available, and they're $10,000 or more.

As LCD displays grow in size, it's exponentially harder to make them, because of the sharply increasing likelihood of a defect somewhere in the display that will render it useless. There's a saying that, if you pour a tablespoon of sewage into a barrel of champagne, you get a barrel of sewage. Displays are like that – a single defective pixel makes the millions of perfect pixels useless. The prevailing urban legend, probably true, is that nearly half of larger displays come off the production line with defects, and have to be scrapped.

But this is changing. For the last few years, manufacturers have been showing 8-foot and larger LED displays at the January Consumer Electronics Show, and anybody with ten large in their Swiss bank account can get one. It's likely that the price will come down – but still, a decent projector costs less than a thousand bucks. Forecasters have been saying projectors will become obsolete. We'll see.

LCD and plasma displays do have one great advantage: they're bright enough to work in direct sunlight. A projector can't do that, unless it's bright enough for a theater and costs as much as a small house.

LED ARRAYS



Stadium-sized displays take the brute-force approach: buy a million each standard red, green and blue LEDs and assemble an array. This is often called a “Jumbotron,” which is not properly a generic term – it's a Sony trademark. Sony introduced the first one in 1985. It used multiple CRTs instead of LEDs because LED technology wasn't ready in 1985. Ironically, the company that invented the Jumbotron no longer makes them.

You already know the science: take a serial data stream that represents a scanned video image, sequentially read it into several million one-per-pixel control circuits using time division multiplexing, control brightness using PWM. And fork over a small fortune.

Probably the largest LED video display in the world is the canopy over Fremont street in Las Vegas, which is several blocks long. This is the second generation; the first generation used incandescent lamps.

SEGMENTED DISPLAYS

“Video” implies a display where the images can move. There are lots of cases where that much complication is not necessary, where the shapes do not need to move, they only need to switch on and off. I'm going to call these **segmented** displays.

In a segmented display, each segment is not a pixel but a larger form whose shape means something. For example, in a multimeter there will be shapes that say “Volts,” “Amps,” “Ohms,” “Battery” etc in addition to the 7-segment numerals. On a car's instrument panel there may be a fuel-pump shape that lights up to tell you you're low on gas.

LEDS

LED segmented displays originally showed up in clocks, calculators and the like. They were originally used because, at the time, the only other option was incandescent lamps. They're usually placed behind a darkened plastic sheet, like the material used in sunglasses, to insure that non-illuminated segments will be invisible. They're still used where an inexpensive high-brightness display is desired.

LCDs

Segmented LCD displays have all but replaced LEDs. In many ways, they're a step backward: they are not nearly as bright and the difference between light and dark, called the **contrast ratio**, is often marginal especially with low-priced units. However, they have two compelling advantages. First, they draw very little power, which makes them much more practical in battery operated things like wristwatches. Second, they don't have to be assembled segment by segment like LED displays; they're manufactured in photographic batch processes.

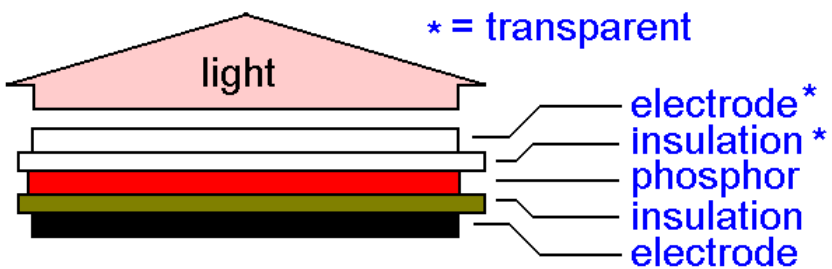
Not all segmented LCD displays have back lighting. **Transmissive** displays transmit light from a backlight source. **Reflective** LCDs have a mirrored surface behind them; ambient light comes in through the face, passes through the polarizer/liquid crystal sandwich, bounces off the rear surface and passes through the sandwich again on the way out. A nice feature of reflective LCD displays is that they can be viewed in direct sunlight.

It is critical that an LCD be driven with the proper signal. When there is no voltage across a given segment, it is off. To turn it on, it must be fed a square wave of fairly low frequency, say 60Hz. It is extremely important that the driving voltage have no DC component, or the LCD will slowly be destroyed.

ELECTROLUMINESCENT DISPLAYS

Strictly speaking, there are many kinds of electroluminescence. (Wow, what a word!) Something that gives off light because it's hot is incandescent; all other ways of generating light electrically, e.g. fluorescent lamps, LEDs, plasma displays etc, are electroluminenscent. However, the term most often applies to a specific kind of display where a phosphor is made to glow – its electrons are excited into a higher orbit, which is the basis of all phosphors' luminance – via direct stimulation by an alternating electric field.

An electroluminescent or **EL** panel is essentially a capacitor, which has two electrodes separated by an insulator. In an EL, there are two insulating layers, one on either side of a phosphor. When an AC voltage is applied across the electrodes, the phosphor glows. Of course, one or both of the electrodes have to be transparent or the glow won't be of any use.



Usually, the phosphor puts out blue-green or ultraviolet light. Other colors can be created by coating the outside surface with another phosphor, which translates the wavelength in the same way the phosphor in a fluorescent tube does.

Typically, the AC voltage is between 90 and 200V. Since power is absorbed by the phosphor, and re-emitted as light, the panel appears electrically like a “lossy” capacitor.

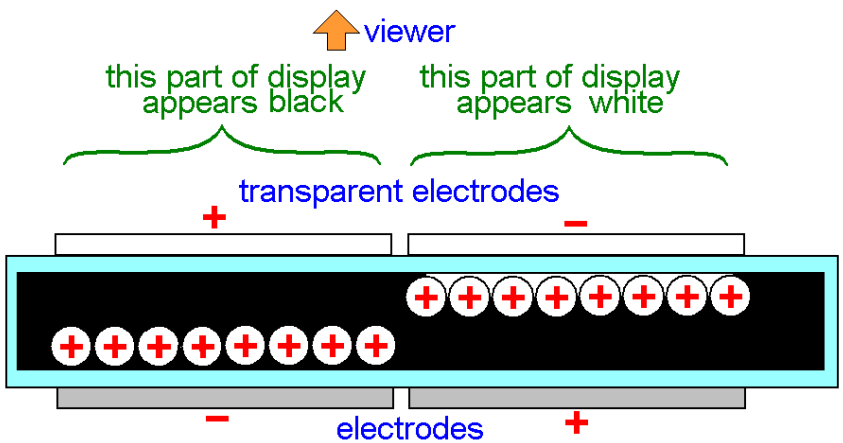
EL displays are efficient and cool-running. Raster-based, pixelized video displays are made by several companies, but have not gained widespread popularity. However, segmented displays are now showing up as advertising and equipment displays. This is nothing new; Chrysler used EL instrument panels decades ago, and then abandoned them. What's pushing their recent rebirth is newer, more flexible control methods, simplicity and long life, and a new emphasis on energy efficiency.

A variation is EL wire, which is flexible and can be bent into any shape. This is a technology you're going to see more of in the future.

ELECTRONIC PAPER

Sometimes, a new technology is so useful that it becomes a big business while the technology itself is still half-baked. Silent films from the 1920s come to mind. The most obvious current example is electronic paper, also called **e-paper** and **e-ink paper**, as used in black & white e-book tablets sold by Amazon, Barnes & Noble etc. There are many variations, and the science is still immature.

E-paper is mechanical, based on the movement of materials. One version consists of two glass panels with a thin space between them, sealed on all sides so the space becomes a chamber. Within the chamber, microscopic particles of white material are suspended in a dark liquid. These particles are charged electrically. Standard metal electrodes on one side of the chamber, and transparent electrodes on the other side (toward the viewer) cause the particles to move to one or the other inside surface of the glass chamber.



When the particles move to the side toward the viewer, the panel appears white; when they move to the other side, the viewer sees the dark liquid and the panel appears black. By dividing the front and back electrodes into pixels, a black & white image can be displayed.

When I say black & white, I mean each pixel is either black or white, with nothing in between – no grays. And to date, attempts to create color e-paper displays have been disappointing; colors are either muddy or washed out. Also, e-paper has much slower response than other kinds of displays, so it can't be used for video. It doesn't even work for smartphone touch screens, because the image lags far behind finger movement.

Despite these drawbacks, millions of e-paper readers have been sold, due to e-paper's two great virtues. First, it looks like paper, it's easy on the eyes, and due to the fact that it's a reflective technology – it is illuminated by ambient light – it can be viewed in direct sunlight. Second, it consumes almost no power. You can use an e-paper reader for months before you have to recharge the battery.

Instead of glass plates, e-paper can be a plastic film with micro-capsules of fluid and particles, and then it can be made almost paper-thin and bendable. I had expected to see billboard-size e-paper displays by now, but apparently the lack of color is a killer in the marketplace. Somebody's going to solve these problems, and then I think e-paper will become at least as popular as all other technologies combined. You read it here first.

PRINTERS

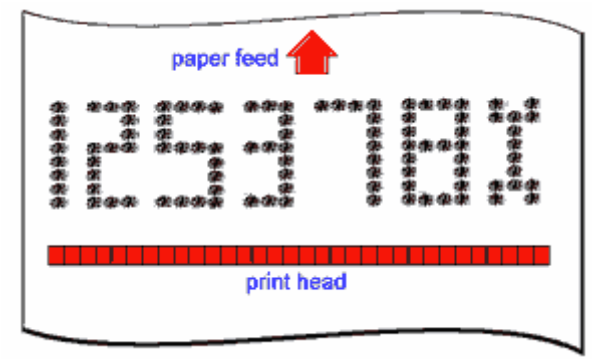
It probably seems strange that I'm lumping printers in with displays, but they actually have a lot in common. A printer produces an image the same way a video display does, by converting a string of serial data into lot of lines that form a 2-dimensional picture. The printer just does it mechanically, with ink. It wouldn't be greatly stretching the point to call the printout a “raster,” and though almost nobody does that, I'm occasionally going to do it in this chapter so you can see the similarities.

Some printers print each line by scanning left to right (or vice versa); others a whole line at once. You could compare these to a serial output port and a parallel port. Speaking of ports, printers used to be the main reason parallel ports existed, but today most printers connect either via USB, or via Ethernet in the case of an industrial-grade printer shared by many computers.

THERMAL PRINTERS

From a mechanical standpoint, the simplest printer in widespread use is probably the thermal printer. In any printer, the part that does the printing is called the **print head**. In a thermal printer, the print head is a heater divided into segments. As the specially treated paper moves by, the segments heat and cool, forming letters in much the same manner as a sign with “moving” lights. There are enough segments to span the width of the paper, and a whole line is printed at once, making this a “parallel” printer.

Thermal printers are simple, reliable, and never run out of ink. Their main drawbacks are that they require special heat-sensitive paper, and they're always black-and-white. They're widely used to print receipts in stores. If you've ever left a receipt in the sun and it's turned black, that was a thermal-paper receipt.



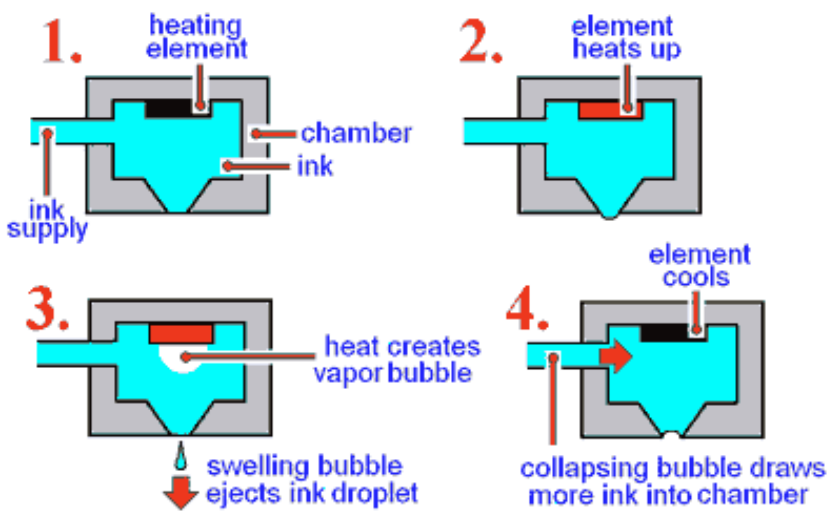
INKJET PRINTERS

An inkjet printer is a “serial” printer; the print head is mechanically moved across the paper, spurting out microscopic droplets of ink with each pass, as the paper is slowly advanced. The ink reservoir travels along with the print head to keep it supplied with ink. In many cases the print head is built right into the reservoir, so every time you get a new ink cartridge, you get a new print head. A plastic belt with very fine lines printed on it, in conjunction with an opto-interrupter, allows the printer to keep track of the print head's location within a few thousandths of an inch. (See chapter 8.) To keep from having to do a separate scan for each line, print heads are designed to print several lines at once.

The biggest technical challenge has not been to get this to work, but to control the amount of ink so the whole brightness range from white through gray to black and the whole color spectrum can be reproduced. Today's printers are the result of many years of development – the basic idea was patented by an amazingly productive scientist named John Strutt, Lord Rayleigh, in 1867. Google him sometime when you're bored.

Thermal Inkjets

Several methods are used to squirt the ink. One of the most common involves a tiny ink-filled chamber with a heating element in it (1 in the illustration below). When the element heats up (2), it vaporizes some of the ink (3). This pressurizes the chamber, causing a droplet of ink to be ejected. When the element cools again (4), the collapse of the bubble sucks more ink in, refilling the chamber. The whole heating/cooling cycle takes only a few microseconds. Generally, a print head will have eight or more chambers for each color, so as to be able to print several lines on each scan across the paper.



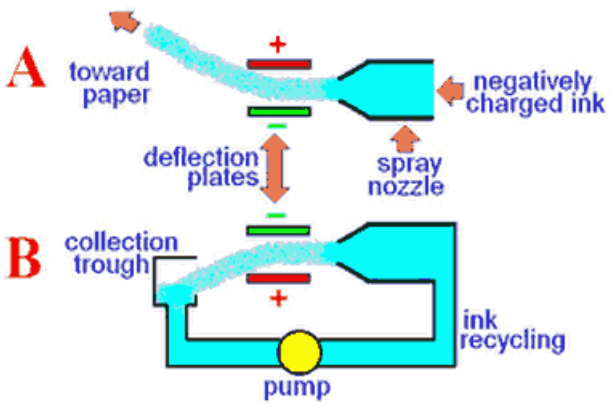
Piezoelectric Inkjets

Piezo inkjets operate about the same way, but they use a mechanical rather than thermal way of spraying ink droplets. You know a crystal oscillator is based on a tiny piece of quartz that physically wiggles in response to an electrical current, at frequencies up to hundreds of MHz. In a printer, a voltage pulse to an ink-coated piece of piezoelectric material causes it to “shrug” and shake a droplet of ink onto the paper.

Continuous Ink Jet or CIJ

In a **CIJ** printer, ink is continually pumped through a high-pressure nozzle. The droplets are given an electrostatic charge, then moved past **deflection plates** whose charge can be switched between positive and negative, bending the droplet stream one direction or another. In one direction, ink hits the paper (A in the illustration on the next page). In the other direction (B), it hits a trough which collects it and routes it back to the pump to be recirculated. Think of a CIJ printer as a DLP projector that uses ink instead of light. CIJ printers are used in high-volume applications like packaging and mass mailings.

Jet printers can print other things besides ink on paper. You've probably seen machines where a moving head prints 3D objects with liquid plastic. This is rapidly replacing the machine shop as a way to make custom forms and molds for industrial casting. I've seen a machine that prints chocolate. In the next few years, we'll see machines that print buildings out of concrete; there are already working prototypes.

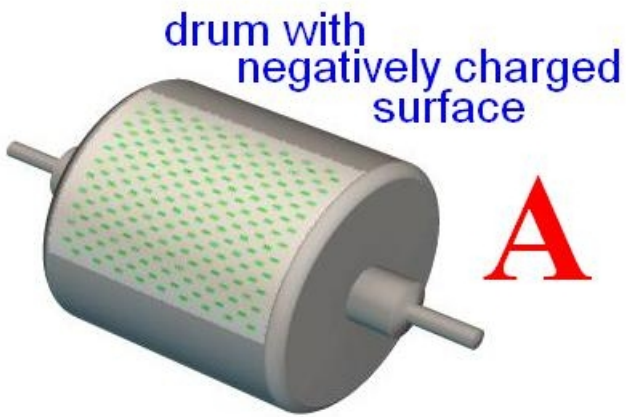


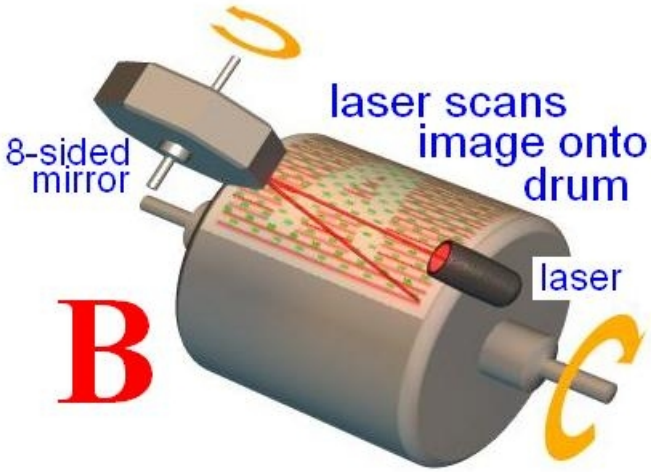
LASER

Light-based printers are based on **xerography**, the technology behind photocopiers. I want to spend a little time on that because a laser printer is nothing more than a photocopier with an electronic input. Besides, you might wind up fixing copiers. You never know.

Let's start with a black-and-white copier. In EM3 we discussed cadmium-sulfide photoresistive cells, which have very high electrical resistance when it's dark but become conductive when light shines on them. Xerography uses a metal drum coated with a similar material. In Xerography, the image to be copied is projected on the drum by optical means. In a laser printer, it's scanned onto the drum by a laser.

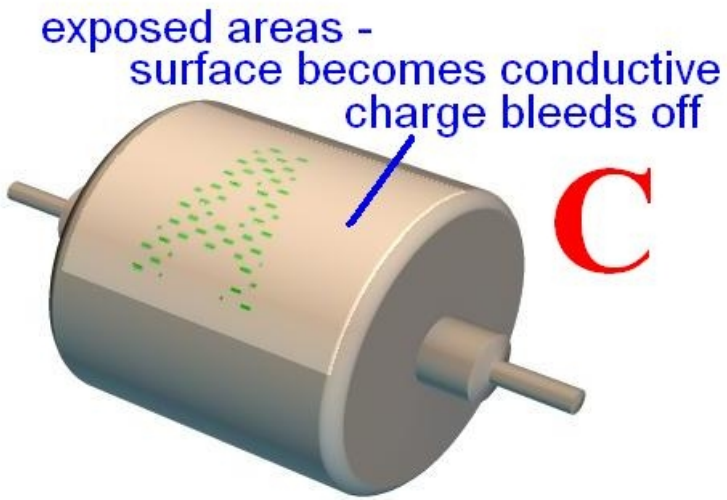
Before the laser begins scanning, the surface of the drum is given a negative electrostatic charge ("A" below, negative charge represented by green dashes). At this point, there's no light, so the photoresistive material is in a high-resistance state and the negative charge stays on its surface.





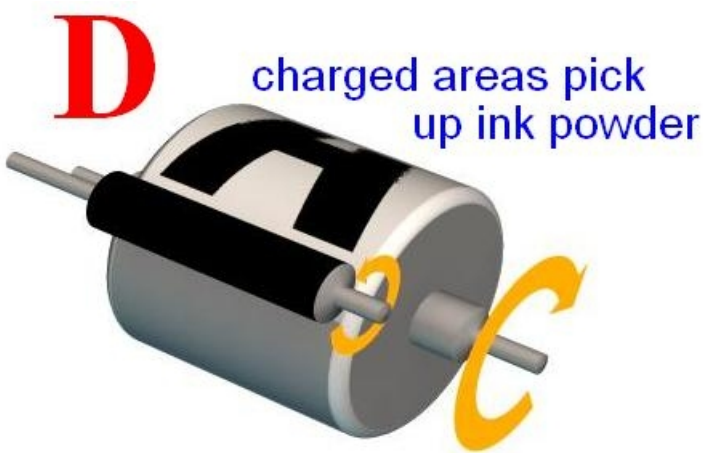
Next, we transfer the image to the drum. The laser is reflected onto the drum via a spinning, 6- or 8-sided mirror that causes the beam to rapidly scan across the drum (“B”). At the same time, the drum is slowly rotated so each successive line is slightly displaced from the previous one. I'm sure you'll recognize this process as scanning a raster, just as an electron beam scans a raster in a CRT, and of course the laser can be turned on and off for light and dark parts of the image.

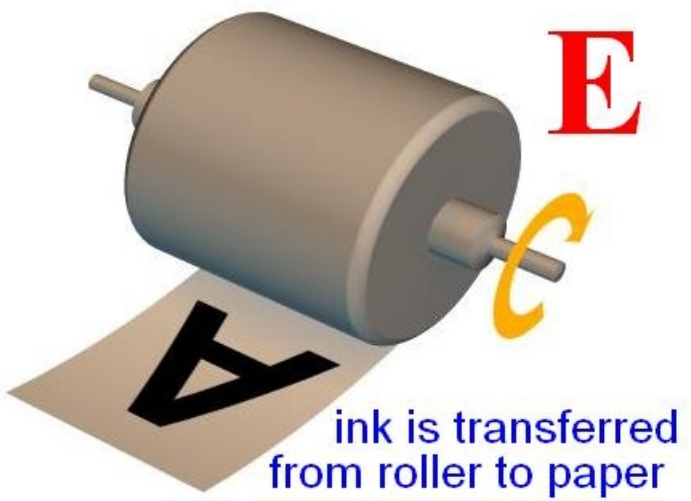
In the light parts of the image, where the laser hit the drum, the photosensitive material goes into a low-resistance state. This allows the negative charge to drain off to the metal drum. But in the dark parts, where the laser was turned off, the negative charge remains (“C” above right).



Then we bring the drum in contact with powdered black ink, which we call **toner**. Where there is negative charge, the toner sticks. Where there isn't, it doesn't. This creates a black ink image on the drum (“D,” next page).

Paper is then brought in contact with the drum, and the ink is transferred to the paper. In some printers, the paper is given a positive charge to encourage the ink to move from drum to paper (“E,” next page).





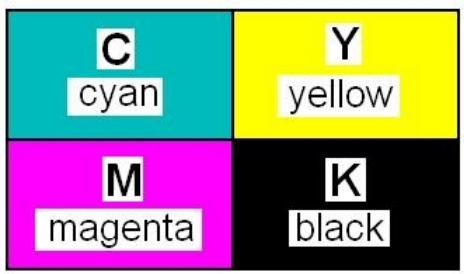
Finally, a bright light heats the ink, melting the plastic and baking it onto the paper. With the microscopic ink particles involved, this can happen literally in a flash; a xenon flash tube is often used as the light source.

Of course, if we carefully modulate the brightness of the laser, instead of just turning it on and off, our image can have the whole range of tones from black to gray to white. This is a tricky process, but they've had half a century to work on it, and it's pretty well perfected by now.

COLOR PRINTING

Just as a color video image is assembled from three primary-color images, a color printer is several printers in one. For comparison, video is an **additive** color process; the individual colors add light and the sum of all of them is white. Printing is a **subtractive** process; each color subtracts (absorbs) light and the sum is black. As previously stated, in a video display the three primary colors are red, green and blue. In a subtractive process, their opposites are used: **yellow** (the chromatic opposite of blue), **cyan** (the opposite of red) and **magenta** (the opposite of green).

Although those three inks together make black, it takes a lot of ink – reasonable amounts of ink just result in a muddy gray-brown. To get a good, dark black, a fourth ink "color," dark black, is added. The resulting system is called the **CMYK** color system, with "K" being black. In color printing, there are four separate images that must be perfectly aligned with each other to make a color picture. With video this is called **convergence**, with printers it is called **registration**. Black is "K" because it's the key, the reference that the other colors are aligned to.



In an inkjet printer, the four reservoirs and four print heads are lined up next to each other. The firmware in the printer's MCU delays the data feed to three of them by precisely the right amount so the four color images register properly.
In a laser printer, four separate images must be laid down on the roller. This means the roller must be four times as large as the paper, or four rollers must be used, or one roller has to make four rotations per sheet of paper, receiving a new image with each rotation. In each case, images are transferred to the paper one color at a time. Alternately, the four images can be transferred to a composite image on a wide belt, which then transfers the image to the paper.

RESOLUTION

There's an important difference between the way resolution is specified for video displays vs. printers. In video displays, resolution is usually specified *for the whole screen;* i.e. 1024 horizontal pixels across the display, from the left edge to the right edge.

In scanners and printers, resolution is measured *by the inch*. They don't use the word "pixels," but rather **dots per inch**, abbreviated **DPI**. Dots and pixels aren't directly comparable, but it is a fact that cheap printers routinely achieve resolution that is only beginning to be approached by the most advanced video displays. It's amazing that they can get that rather clumsy color printing process to work as well as they do.

One more chapter to go. You've learned a lot about digital circuits and equipment, now it's time to take a quick look at the more common techniques for servicing them.

End of chapter 9

EM4 Chapter 10
DIGITAL SERVICING TECHNIQUES

Objectives of this chapter:

- *Describe the following and explain the problems they cause:*
  - *noise*
  - *transients*
  - *ringing*
  - *high frequency attenuation*
  - *reflections*
  - *crosstalk*
- *Define and discuss "skew" and "jitter"*
- *Define and discuss "ground bounce" and "rail bounce"*
- *Explain the function of a logic pulser, logic probe and current probe*
- *Discuss sweep-type and real-time (FFT) spectrum analysis*
- *Describe the functions of a typical digital oscilloscope*
- *Describe the functions of a data analyzer, and what the four displays represent*
- *Discuss test points and where they would be found in equipment*
- *Explain the use of repeaters*

"Digital servicing techniques" means both servicing digital equipment, and using digital test gear to service analog equipment. I want to begin with a couple of cautionary topics. Keeping these in mind will save you endless hours of frustration and shorten the time between when you start to do digital servicing and when you master it.

First, I've said this before, but it applies with double force when using digital equipment to service analog gear and triple force when using digital to service digital: don't take measurement results at face value. It may be comforting to see a nice, precise number on a display – or it may be aggravating, if the number isn't what it's supposed to be – but in neither case can it be assumed that the number is actually telling you something real and useful.

Bruce Archambeault, PhD, commented on this in the June 26, 2008 issue of Electronic Design News, and I saved the article because it perfectly describes what I've learned from my own experience:

"Too many engineers [and techs] ***just accept whatever they see on a measurement device as gospel. In reality, every measurement provides ample opportunity for mistakes. Your probe may load down the circuit, you may fail to use an appropriate reference connection – that is, ground connection – or fail to account for a***

changing antenna factor when operating near a metal floor. I firmly believe that measurements are important, but you must question everything – both measurements and simulations – to ensure correctness...

"Every tool gives accurate answers to the questions you put to it, but there is a catch. You must know what questions to ask...Lacking that understanding, no tool can create useful models, but every tool will give impressive-looking answers. That's the pitfall.

I would add that with digital we're often sampling and quantizing analog values, and that presents additional pitfalls. We'll be talking about some of those.

Second, remember that everything's really analog. When data is being sent over a cable, we usually don't call it "signal," but that's what it is. It's still voltages and waveforms. Very often, when 1s and 0s aren't what they should be, it's because the analog signals that represent them have been loaded down, smeared, glitched or mixed together beyond the digital circuitry's ability to correct.

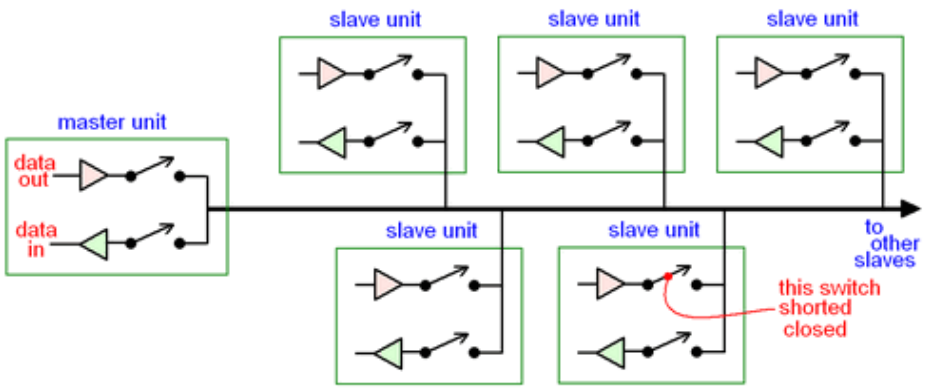
Tell you a story: I was once called to fix an RS485-based data collection system in a factory. An RS485 system consists of one master and a number of slave units all connected to a common data cable. In a sense, it works like a data bus inside a computer. The difference is, the bus is a multidrop data line and the various nodes are spread all over a large building such as a factory or warehouse. The master unit polls each slave in turn, asking "Have you got any data to send?" It then releases the line (switches to receive mode) and waits for an answer. If no answer is forthcoming, it switches back to transmit and polls the next slave in sequence. However, if a particular slave does have information to send, that slave switches to transmit mode, seizes the line, and sends its data to the master.

I first encountered RS485 when it was relatively new, and I didn't know anything about it. In the malfunctioning system I was called to diagnose, some units didn't work reliably, and when replaced with new units, the new ones didn't work reliably.

So I got out the scope and watched what happened on the line, and eventually I figured out the polling-and-response business. I also noted that the farther a slave was from the master, the weaker (lower voltage) its signal – *except* for one unit that always answered full strength, and always worked correctly. On a hunch, I unplugged that unit. Voila! All the other units started working.

Why would the one unit that always worked perfectly be the source of the problem? Well, in early RS485 systems, the common data cable was connected directly to the CMOS ports of each unit.

And, of course, each unit was locally grounded, as required by the National Electrical Code. So there were often massive ground potential problems, and in this case one of those CMOS ports was damaged. It still switched between send and receive, but it wouldn't switch into high-Z mode, meaning it wouldn't release the line. Its loading attenuated the data from the other units just enough that they would sometimes work, sometimes not.



Moral of story: the problem had nothing to do with anything digital. It was plain old analog attenuation of voltages, a voltage divider made of the shunt resistance of the defective unit and the series output resistance of some other unit. **Most problems that occur in the communication links between units are analog in nature**.

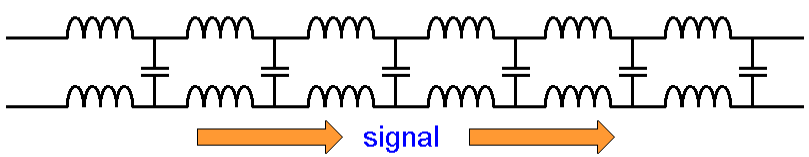
Of course, in many cases a malfunction is an out-and-out failure of a component or connection; something is just broken, doesn't pass a signal, doesn't work at all. Those are the easy ones to find and fix. The tough ones are intermittent malfunctions. These can be roughly divided into two categories:

- a component or connection that is intermittently bad, e.g. a cold solder joint that sometimes makes and sometimes doesn't. A good way to find one of these is to tap sharply on various components with an insulated tool (e.g. plastic rod) while watching a scope trace or, in the case of audio, listening, and in the case of video, watching. Also, a service tech should keep a bottle of freeze spray handy. Spraying one component at a time, sharply lowering its operating temperature, often reveals an intermittent component.

- a marginal component, equipment design flaw or interconnection problem that degrades the signal until it's just on the edge of not working. In digital equipment especially, there's usually a fairly well defined division between “good enough to work” and “not good enough.” A system that's on the edge may slip in and out of proper operation, or may operate properly but with a much higher than expected bit error rate. This can occur inside a unit or in the cables and interfaces that join units together.

Here are a few of the more common causes of signal degradation that you'll run into.

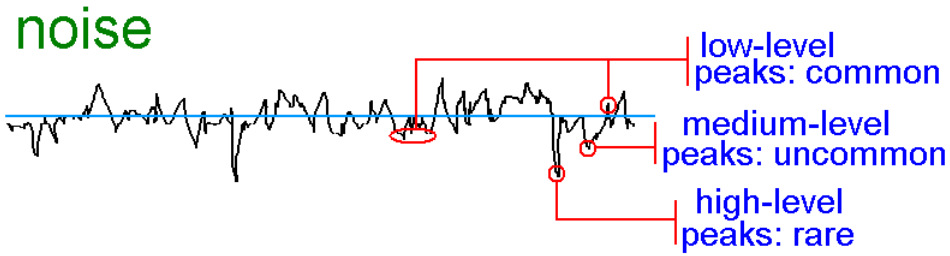
Attenuation. A cable is, essentially, a series inductance and a shunt capacitance along its length. If we imagine these reactances chopped into lumps, we see that what we've got is a many-stage low pass filter.



A given type of cable has a given attenuation per foot at every frequency, and at high frequencies, the attenuation is severe. A signal going in one end at a volt may come out the other end at millivolts or microvolts.

This by itself isn't necessarily a problem, as the weak signal can be amplified at the receiving end. However, the receiving unit has limited amplification, and if the signal drops low enough, the system won't work. Also, cable attenuation leads to other problems discussed below.

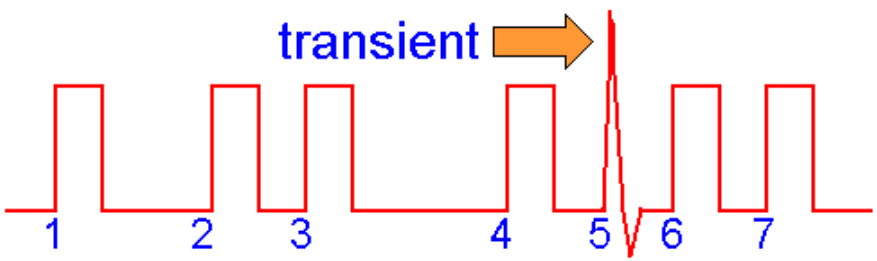
Noise. As noted in EM3, every system has a certain amount of white noise. The instantaneous noise level is random, varying moment by moment, with the highest-voltage peaks occurring least often.



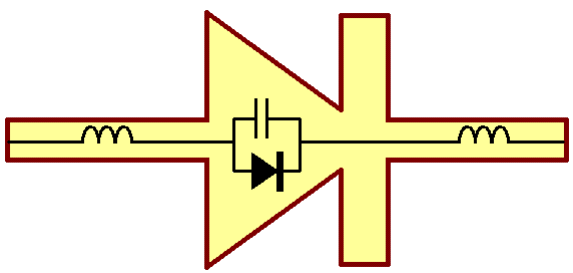
If a desired signal is weak enough, the highest noise peaks will overwhelm it, creating an occasional data error. If the signal gets still weaker, a higher percentage of the noise peaks will overwhelm it and the data error rate will rise. At some point the data will become unusable; even before that happens, the data transmission rate will slow down as more data needs to be re-sent.

Increasingly, data is transferred wirelessly, for example WI-FI hotspots. Wireless data links routinely deal with very low power levels and signals at the receiving antenna so weak they are in danger of falling below the noise floor.

Transients. Besides background white noise, the environment contains random transients that are many orders of magnitude higher in intensity. If they are as large as the 1s and 0s, the system may count them as extra bits.



It's almost impossible to completely suppress the transient that occurs when a current-carrying circuit is broken. In EM3, we saw how a diode can be connected backwards in a DC circuit to snub transients, but in the case of very high-speed transients, even a diode has limitations because its parasitic junction capacitance, together with the inductance of its leads, makes a resonant circuit.

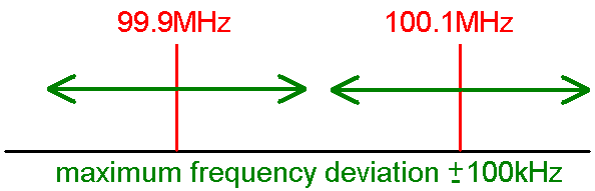


Like artillery, transients can come flying in from any direction. Since you can't always control them at the source, the ideal solution is to design the system so it is immune to received transients. For this to happen in a large system with many locally grounded units, the data lines would have to be *both* balanced, like Thunderbolt, *and* isolated, like modern RS485. As far as I know, there is no such existing standard. The other way to achieve immunity is to use optical fiber.

Differential Attenuation and Delay. As noted, cable attenuation increases with frequency. In addition, a cable delays the signal. The distributed inductances and capacitances in the cable equivalent LC diagram above amount to a **delay line**, and in fact short analog delays are traditionally achieved using discrete inductors and capacitors in exactly this fashion.

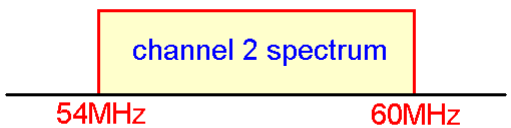
If a signal occupies a narrow frequency band, this differential behavior – higher frequency signals attenuated and delayed more than lower frequency signals – doesn't matter.

For example, an FM radio occupies a 200KHz-wide channel at about 100MHz, which is to say its lowest and highest frequency limits are about 0.2% apart.

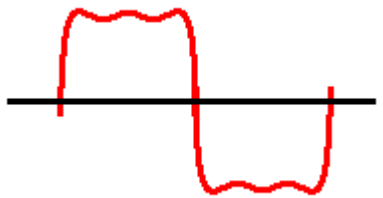


Whatever effect a cable may have on an FM signal, it will affect all parts of that signal about equally. So the signal will emerge from the other end attenuated and delayed by a few microseconds, but otherwise unmodified. In fact, since the entire FM band exists within 10% either side of 98MHz, the attenuation differential across the whole band will be minor.

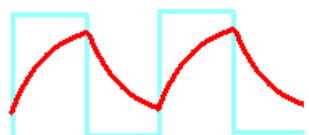
The situation is the same where multiple signals occupy a wide band, but each signal is narrow-band. For example, many cable TV systems still distribute analog signals on the old analog channels 2-13. The higher channels are attenuated much more than the lower channels, but differential attenuation within each channel is minor. The worst case is channel 2, which is spread over 6MHz of spectrum at approximately 60MHz, or a differential frequency span of around 10%.



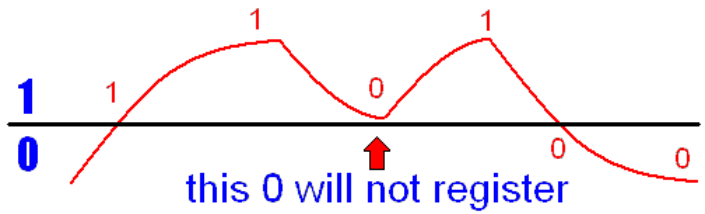
With digital data the situation is different. A data stream usually takes the form of square waves, and as discussed in EM3, a square wave is a composite of many harmonically related frequencies. For a waveform to be a square wave, even a sloppy one as shown here, third- and fifth-harmonic energy must be present. That means the signal occupies at least a 5:1 bandwidth; for a 100MHz square wave, harmonics to at least 500MHz.



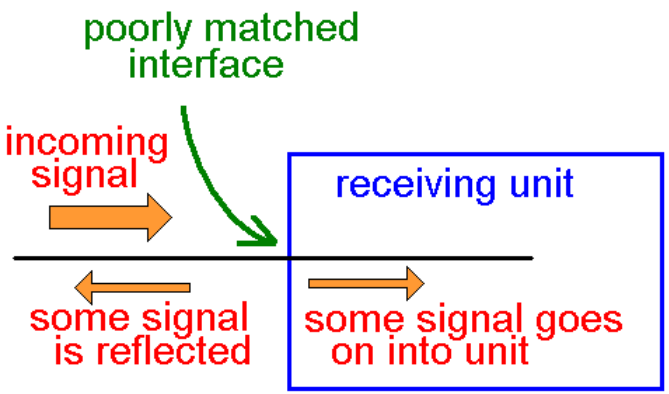
In this situation, time delays and high-frequency attenuation cause some aspects of the signal to be attenuated and delayed much more than other aspects, changing the waveform. In the illustration at right, which shows a situation that is not at all uncommon, the original wave (light blue) has been smeared (red) to the point where some circuits would have a hard time telling where the waves start and stop.



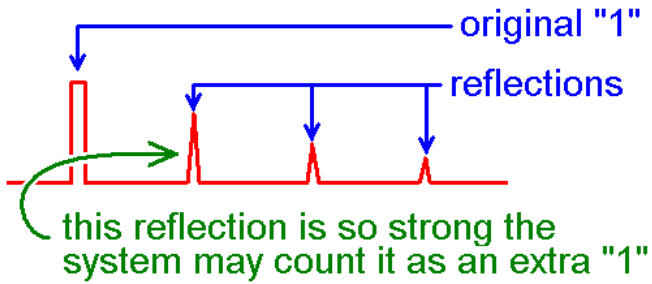
At worst, the transitions can be so rounded off that the voltage doesn't have time to move past the transition point between 1 and 0, causing bit errors.



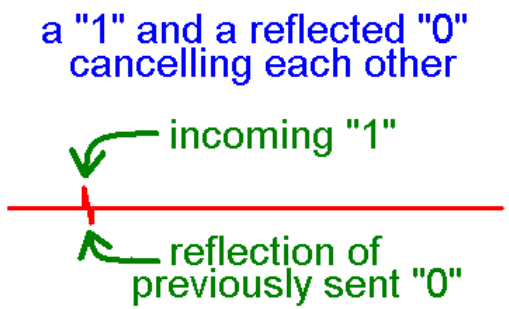
Reflections. As you know, a transmission line should be terminated at each end by a source and load whose own impedance is the same as the line's impedance. When this is not done, part of the signal will be reflected back toward the source.



If the source is also poorly matched, which is often the case, the signal will bounce back and forth like echoes. If the echoes are strong enough, the receiving unit may count them as extra 1s and 0s.



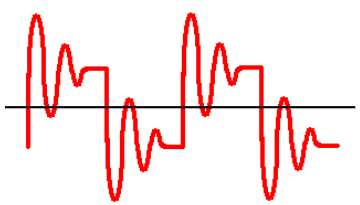
Or, if they are out of phase with new data coming down the cable – if a fresh, first-time data 1 arrives at the same time as a previously sent, reflected data 0 – they will cancel. (This is called **destructive interference**, and we'll deal with it a lot in EM6.) In either case, data at the input of the receiving unit is corrupted.



When the signal is spread over a wide band, as with digital, termination is not as simple as picking the right resistor. A cable's characteristic impedance varies significantly with frequency, so no single source or load impedance will properly terminate it. Some reflections are guaranteed.

(One way some high-speed systems deal with this is to deliberately rely on high attenuation in the cable to weaken reflections.)
The proper termination impedance for an RF cable is 50 ohms, or 75 ohms if it's carrying video information. Unshielded twisted pairs with the standard white/blue, white/orange, white/green etc color code are usually 100 ohms.

Ringing. The higher the data rates, and the higher the frequencies, the harder it is to avoid resonant LC circuits. Every connector, every PC trace, every IC socket is a resonant tank, or many tanks. The initial resonant peaks resulting from a square wave transition can be strong enough to count as extra 1s and 0s.


square wave with severe ringing

Crosstalk. At high frequencies, every capacitance is a significant capacitance and every inductance is a significant inductance, and equipment designers have to go to heroic lengths to keep signals from coupling into each other. Any 1-to-0 transition on one connector pin can easily become a 1-to-0 transition on the adjacent pin, even though there's no connection between them.

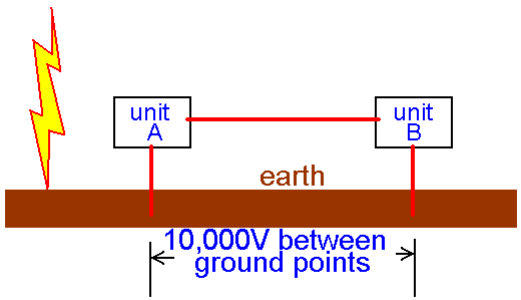
Balanced pairs help, but they don't completely solve the problem. Twisted wires have to terminate in straight connector pins, and at 10GHz a pair of connector pins can be a tightly coupled system. Things like Thunderbolt connectors are carefully laid out to cancel as much mutual coupling as possible.

The actual length of even short wires and PC board traces can be critical. In EM6 we will talk about the fact that a transmitting or receiving antenna has a resonant frequency that's related to its length, and an antenna is tuned by adjusting that length. At 10GHz, every PC board trace has to be checked to be sure that it isn't the resonant length of any frequency it will be handling.

I know, that's an engineer's job, not yours. But not every engineer does their job perfectly, and techs often wind up tracking down problems that weren't fixed in the design stage. Also, even if the various units are properly designed, problems can arise with the interfaces between the units.

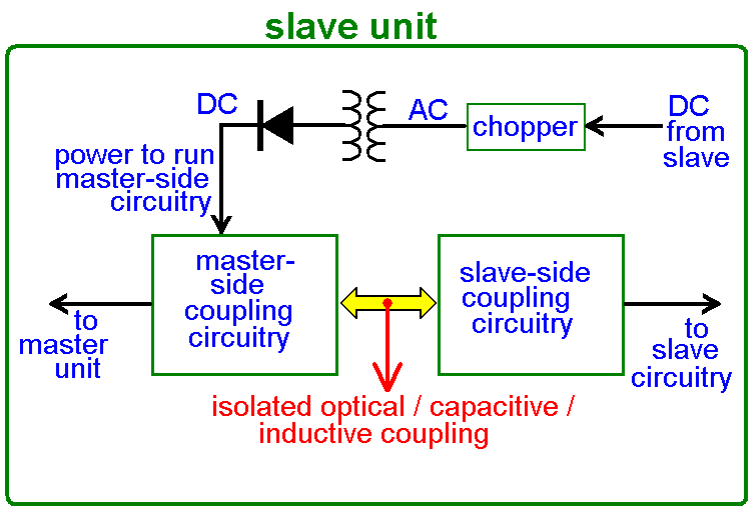
Ground Potential Problems. One of the sneakier sources of transients is ground-voltage differences. Just because two units at opposite ends of a factory are locally grounded at their power outlets, that doesn't mean their frames or the ground pins of their connectors are at the same voltage.

All sorts of natural and artificial electrical disturbances in the area – a lightning strike, for example – can cause widely separated grounds to be at different voltages.



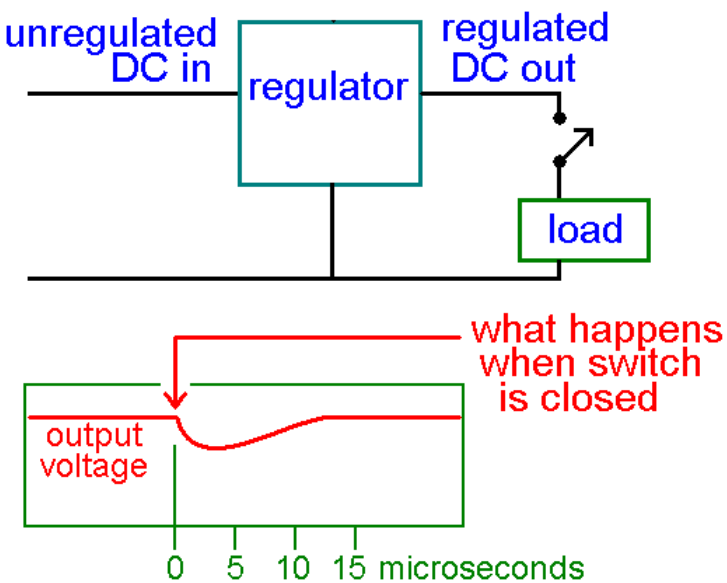
The same thing is true of computers on the 50th floor of an office tower. Electricity travels at 3 x 10^8 meters per second, and a 50-floor building might be 300 meters tall. If lightning strikes the building, it must travel down the structure for a microsecond before it hits a real ground. In the meantime, the office and every ground in it and everything connected to every ground are going to go to 10,000 volts for an instant, and there isn't a thing anyone can do about it. This is an extreme example of what is called **ground bump**. It was ground bump that destroyed the output gate of that RS485 unit I was called in to service, and I wasn't a bit surprised when I began to see manufacturers offering RS485 isolation units in DIP packages.

The illustration below shows how an RS485 slave unit handles isolation. On either side of the interface (yellow arrow) there is circuitry that converts from a wire connection to an optical or other electrically isolated connection. The RS485 multidrop data line does not provide power to run the master-side coupling circuitry because the need for isolation was not understood when the standard was established. So, the slave provides it. DC is chopped up into AC, fed through a transformer to achieve isolation, then rectified back to DC.



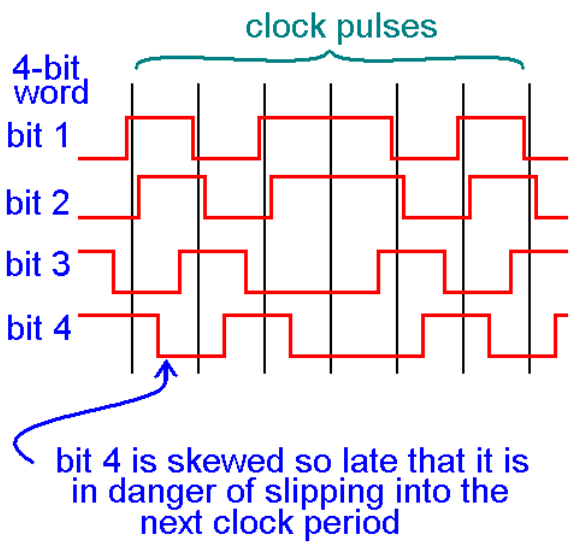
In chapter 6, when I told you RS232 uses up to +25V and -25V for “0” and “1” respectively, did it strike you as odd that the voltages would be so high? One of the reasons is that they're trying to stay above the large ground potentials often encountered in industrial buildings.

Whether equipment based on Ethernet and other standards has ground bump protection depends on the specific piece of equipment. Of course, fiber optic links do not suffer from electrical ground potential problems. Be aware, though, that a fiber cable sometimes includes DC power wires too.



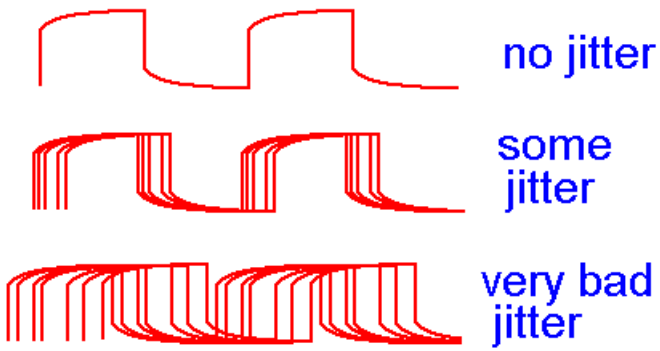
Power Supply Variations. If the DC supply voltage is unstable, circuit behavior will be unstable. With regulator chips as common as they are, why should a power source ever be unstable? Because regulator chips take a few microseconds to respond to changing loads, and because they're not usually right at the point of load, especially when power is part of a data cable as in USB. This could be called **rail bump**, shown below left.

Skew. This refers to timing inconsistency between one bit and the adjacent bit in a parallel word. If all bits do not arrive at the same time, and the differences are great enough to cause uncertainty about which clock pulse a given bit corresponds to, a bit may wind up in the wrong word. The higher the data transfer rate, the more serious skew becomes.



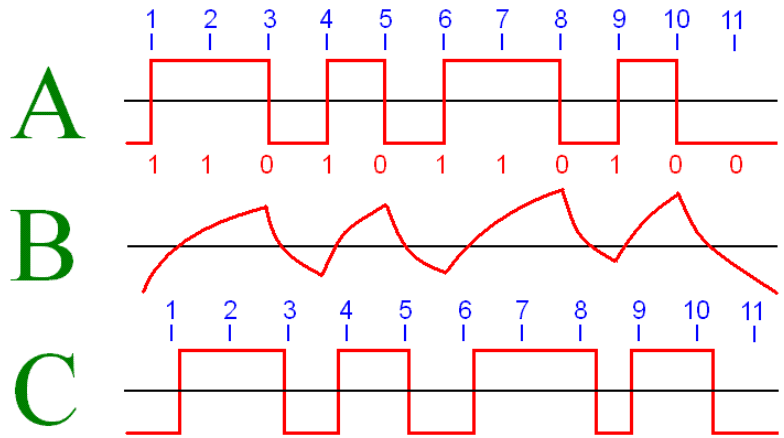
Jitter. This refers to timing inconsistencies in the timing of 1-0 and 0-1 transitions. Suppose we have a 10MHz square wave, which means a transition should occur every 100 nanoseconds.

Suppose our measuring equipment tells us that actual transitions are occurring early or late by as much as 1 nanosecond. That means the timing has a peak jitter of 1%.



There are many sources of jitter. The oscillator providing the clock always has some. If a CPU is processing a data stream from somewhere else that's synchronized to a different clock, there will be uncertainty about which local clock cycle each data transition will fall into. One of the duties a CPU often has to perform is to re-time jittered data, harmonizing transitions with the local clock.

Even if a data stream with no jitter could be created and fed into a cable, it would have jitter by the time it got to the other end. Consider the illustrations to the right, where the blue numbers indicate clocks and the red numbers indicate 0-to-1 and 1-to-0 transitions. At "A," perfect, jitter-free data, perfectly aligned with the clocks, is sent into a cable. At "B", badly smeared data comes out. Note that so much integration is taking place (if you haven't done EM3 yet, you'll learn about integration when you do) that the waveform doesn't have time to go fully up or down between transitions, but it moves farther when there are two 1s or 0s in a row. This shifts the average DC voltage.



At "C," the receiving unit has squared off the waveform again. There is an overall delay – the clocks are shifted – but that is not usually important. What's important is that the individual transitions come earlier or later than they are supposed to.

Thus data that comes out of a transmitting unit perfectly timed will look to a receiving unit like it has jitter, simply because it's gone through a cable.

Modern technology has made the jitter problem worse. I mentioned in a previous chapter that serial data links are cheaper than parallel, but there's an additional reason for using them. Because of skew, at the highest data rates it's difficult to make a parallel bus work if it's more than a few inches long, which means even within a motherboard it's an issue. Serial data solves this problem; all the data bits and the clock are integrated into a serial stream, and anything that skews the timing skews it equally for all components of the signal. This is true even of the four-lane Thunderbolt, which looks like a four-bit-wide parallel data bus. It's not. The four lanes are four separate serial data links running side by side.

So serial transmission solves the skew problem. But the bit rate of serial data is at least eight times higher than the equivalent parallel rate, usually more, and this makes jitter more of an issue. When the data rate is 100Mb/S, each bit takes 10nS and 10pS of jitter, 0.1%, hardly matters. At 20Gb/S, each bit takes 50pS and 10pS of jitter is a 20% timing error, enough to start causing problems.

(In the description above, I'm being a bit careless. There are various ways of packing bits, e.g. using more than two voltage levels, that reduce the actual frequencies involved. However, all the highest-speed data systems use these techniques and push the data rates as far as skew, jitter and attenuation allow.)

Which of these problems a tech needs to look out for depends on the situation. For example, if you're working in a factory where machines are digitally controlled, you aren't likely to be dealing with high data rates where jitter is a problem, but you'll have to deal with ground bounce.

Of course, we shouldn't overlook plain old circuit malfunction, like a burned-out IC. In fact, that's where we're going next.

ESD PROTECTION

We're going to start talking about digital test equipment and troubleshooting techniques, and I want to begin by alerting you to **electrostatic discharge** or **ESD**. ESD is what we call “static,” the electricity that builds up when you shuffle your feet across the carpet. ESD can come from anywhere – a shirt made of flannel or synthetic material, rubber shoe soles, cleaning rags, and a million other things.

There's a widespread belief that a very dry environment poses a greater ESD danger than a humid one. There is a little truth to that, but it's about like the difference between boiling water and almost-boiling water: they're both so hot, the difference doesn't matter much. The best bet for a tech is to assume ESD risk is determined by the static-generating materials present and by grounding, and not humidity.

When we deal with digital equipment, we're almost always dealing with circuitry that's full of MOSFET transistors and ICs. MOSFETS have open-circuit inputs which are very vulnerable to voltages that exceed their specified limits. Those limits tend to be in the 5V to 15V range. ESD tends to be in the 1kV to 10kV range. That's a big potential (pun intended) problem. As I've mentioned before, there are few things as embarrassing as breaking the equipment you're supposed to be fixing.

It should be second nature to techs who work with digital equipment to Earth-ground everything in sight. You should wear a wrist strap that connects you to ground through a 1-megohm resistor located near the strap. The resistor provides a good enough current path to drain off any static electricity you may be carrying, but it's a high enough resistance to limit current to a safe value, should you be accidentally connected to an AC line voltage source. Don't use a wrist strap connected directly to ground; that's an electrocution waiting to happen.

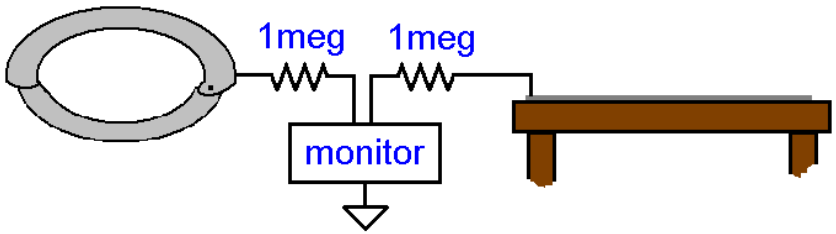
Most bench tops are insulators – wood, plastic, vinyl, painted surfaces etc – where static can easily build up. If you work with digital stuff all the time, you should cover the bench with a metal plate, grounded through a 1-meg resistor. Otherwise, you should have an anti-static mat that you can place on the bench, similarly grounded. The 1-meg resistor should be near the benchtop or pad, not at the other end of the wire.

It goes without saying that you should use a grounded soldering iron. This is a bit tricky, because the ground means you may occasionally short out a voltage that's lurking in the unit under test. The cure for that is to be sure everything in the circuit is discharged before you apply the soldering iron.

After about the third time you blow something up with the grounded iron, you will be tempted to insert a ground-lift adapter. Don't. A lot of the AC voltage in the iron gets capacitively coupled to the tip. This is especially true of variable-temperature irons, which use triac switching to control the temperature. Ground that iron.

You might ask, “Who cares if the iron is grounded, the chip I'm removing is already blown?” Sure, but it's connected to other chips around it. If you keep blowing up good chips in the process of removing bad ones, you will never get the unit working again.

You may want to invest in a ground monitor, which continuously tests the wrist strap and bench top to make sure the ground is good.



DIGITAL TEST EQUIPMENT

I'm going to describe the simpler pieces of test gear as if they were independent, stand-alone units. This is often not the case nowadays; these functions are usually integrated into more comprehensive test gear. But the functions themselves are still basic to digital servicing. I won't go over digital multimeters, since they simply represent a digital method of measuring analog values, and you presumably studied them in EM1.

We're going to start with traditional techniques and then move forward to the present day. This means we'll be dealing first with component-level servicing. I've been told this skill is obsolete, that people just swap out a whole PC board and send the defective one back to the factory.

I'm told the tech only needs to understand things at the system level. There is truth in that, but it's also true that, once those defective boards get back to the factory, they are not necessarily scrapped. In many cases, they are repaired, and that takes techs. You never know where your career will take you.

It's also true that not all digital equipment lives at the cutting edge of new technology. You might want one of the jobs where component-level servicing skill is helpful. And, deep knowledge of electronic theory and component-level behavior is an asset when you're trying to diagnose a system malfunction.

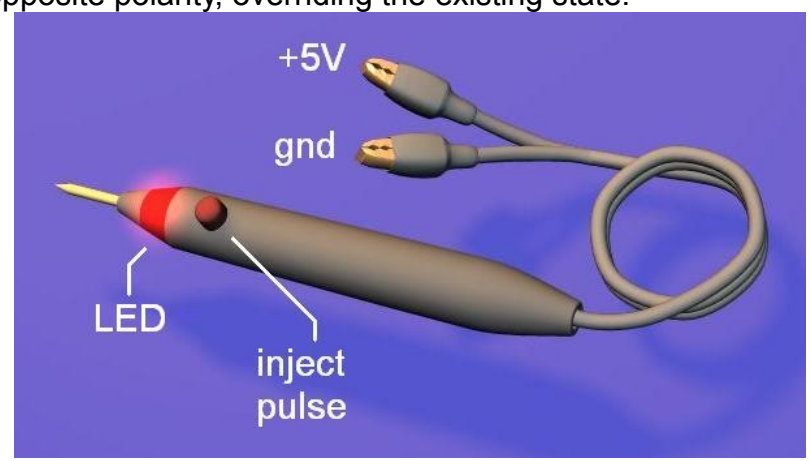
We can roughly divide test gear into output devices, which create test signals, and input devices, which look at them. In some cases a tech uses an input device to look at existing signals, trying to track down a recurring or intermittent glitch in an existing data stream. At other times, a tech uses an output device to inject a known signal into the system at a particular point, and then attaches input devices at various other points to study what the system does to the known signal. The advantage of this approach, when it's possible to use it, is that you know exactly what the signal is supposed to look like.

More advanced test gear both generates and analyzes signals. This is often an advantage because, whens signal generation and analysis are coordinated, it's possible to generate and analyze much more complicated data patterns that test the system much more thoroughly.

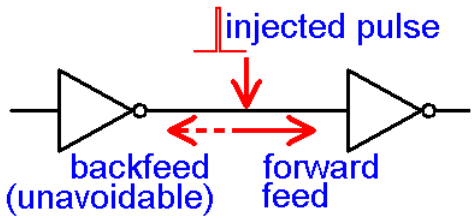
LOGIC PULSER

The easy way to tell whether a problem is occurring "upstream" or "downstream" from a certain point in the circuit is to inject a known signal at that point.

The logic pulser is an output device that generates this signal. When you attach the pulser to a point in a circuit, it senses whether the existing state is a "1" or "0." Then, when you activate the pulser, it puts out a pulse of the opposite polarity, overriding the existing state.



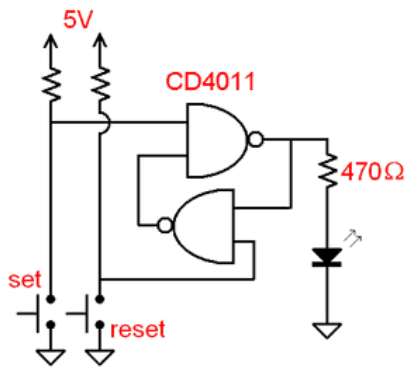
In doing so, it usually backfeeds a logic output, forcing it into the opposite state. You'd think that would destroy the output, but the pulse is only a few microseconds long, not long enough to overstress the circuit. In the example shown on the previous page, an illuminated LED indicates that the existing logic level is 1.

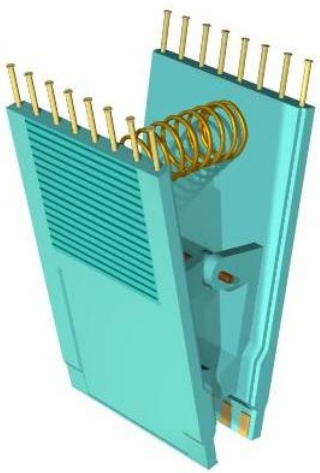


LOGIC PROBE

An input device, but not the kind of probe used as an input device to an oscilloscope. A logic probe is a self-contained unit used to determine whether a particular point in a circuit is "0," "1" or switching back and forth. The simplest probe looks like the logic pulser shown earlier. The logic state is indicated via an LED: "1" makes the LED light, "0" makes the LED go out, and data toggling back and forth between 0 and 1 makes the LED flash.

The LED is almost never driven directly by the circuit under test, because it might load the circuit enough to change its operation. For example, if a 470Ω resistor and LED are attached to the output of the SR flipflop as shown at right, it probably won't set; the upper NAND can't pull up hard enough. As usual, the number one commandment of testing applies: Thou Shalt Not perform the test in such a way as to upset what's being tested. The logic probe must either have its own power supply (e.g. battery) or it must have a clip that can attach to a power supply point in the unit under test.



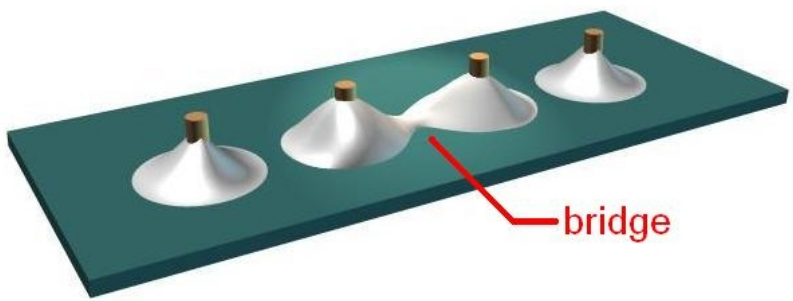


A useful variation for testing older equipment is an extender that clips onto all the pins of an IC at once. Contacts at the bottom are connected to pins at the top, giving you a convenient place to attach scope or data analyzer probes. In newer, smaller ICs, it may not be possible to clip to the leads, and in large MPUs like Pentium processors, the terminals are under the IC and you may not be able to get at them.

CURRENT PROBE

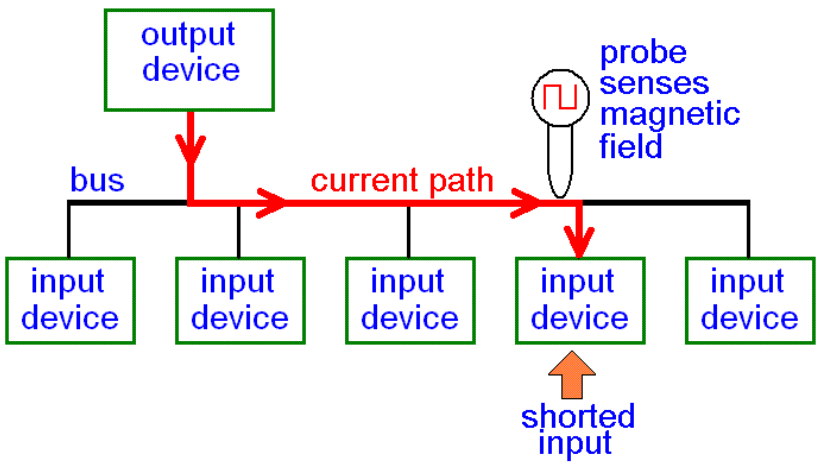
A logic malfunction is often caused by an input or output that is shorted, either because an IC chip has burned out, or, in the case of a manufacturing operation, a solder bridge has been created that shorts a logic point to one of the rails. Because of this hard short, a voltage probe cannot find the problem.

Consider a parallel data bus on a PC board that connects to half a dozen ICs. One of the traces is always “0,” indicating a short to ground. But where is it shorted? Any one of the ICs could be causing the problem; you'd have to remove them one by one, perhaps destroying good ICs or the board in the process. Or the components could all be good but there might be a solder bridge.



A current probe can trace the actual current path. It's an input device, but the input is inductive. When data-signal current flows through the PC board trace from its originating point to the point where it's being shorted out, it radiates a magnetic field which is picked up and amplified by the current probe. This is the same principle used by the clamp-on AC ammeter discussed in EM3.

When the signal gets to the point where it's being shorted out, that's where current flow ends. There's no current flowing beyond that point – not data-signal current, anyway – so the probe will not see a magnetic field. The point where the field ends is the location of the short.



MAXIMUM VOLTAGE

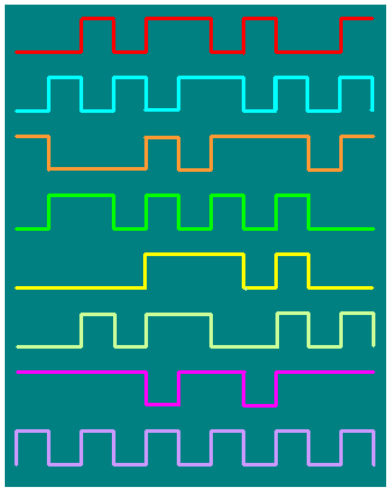
A standard oscilloscope probe can usually handle hundreds of volts, especially if the 10X attenuator is switched in. That's not true of logic probes, which have active circuitry built in. Rule of thumb: never connect an active probe to any point in a circuit where the peak voltage is greater than 15V. Some probes have even lower maximum voltage limits; beware.

Another rule of thumb: don't apply an active probe to an AC power source. Even if its normal peak voltage does not exceed 15V, it may have spikes and noise on it that do. And an AC power source is usually pretty "stiff," meaning that it has a low impedance and is able to force itself into whatever it's connected to. In a battle of wills between an AC power source and an active probe, the probe will lose.

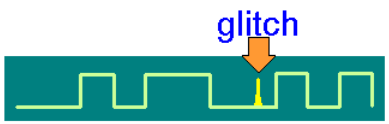
LOGIC ANALYZER

A basic logic analyzer can be thought of as an oscilloscope with 8, 16 or more inputs displayed as parallel traces. Each trace may have its own color to help the tech keep things straight. All the logic analyzer's traces are separately sampled and displayed at the same time, in nearly real time.

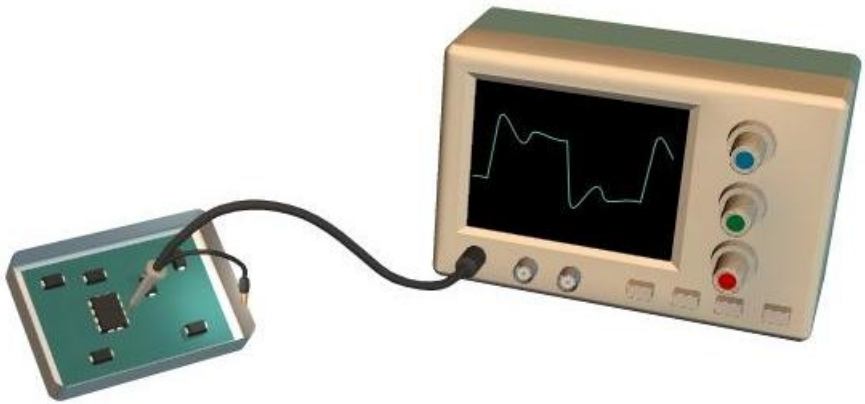
This permits the tech to attach probes at many places in the circuit under test, so the progression of 1s and 0s can be followed through the circuit.



More complex analyzers have additional features; for example, when an unusual event comes along, the analyzer may display it in a different color. “Unusual event” often means “intermittent glitch” which is what the tech is trying to track down.



The inputs to higher-speed analyzers tend to be differential, connected to differential probes – differential inputs are covered in detail in EM3. One thing to keep in mind about digital data is that, since you're dealing with (ideally) steep rise and fall ramps, you're dealing with waveforms containing energy at frequencies far higher than the actual system clock rate. At those frequencies, if you look at a square wave at a certain point, using a ground reference three inches away (to say nothing of three feet, as with a separate ground wire), the waveform you see will bear no resemblance to what's actually going on at that point. An MPU looks at what's going on at one of its input pins in comparison to a close ground, and that's what you have to do.



This has become even more relevant as differential data transfer has become more common. HDMI, DVI, Thunderbolt and a number of other standards use differential data transfer, and you must use a differential input to examine them. So, every time you connect one of the analyzer's inputs to a point in the circuit under test, you must also give some thought to where you connect the reference clip (or pin) for that input.

An additional caution: since a high-speed analyzer probe often has built-in amplifier circuitry, learn the maximum voltage it can handle and never connect it to anything higher than that.

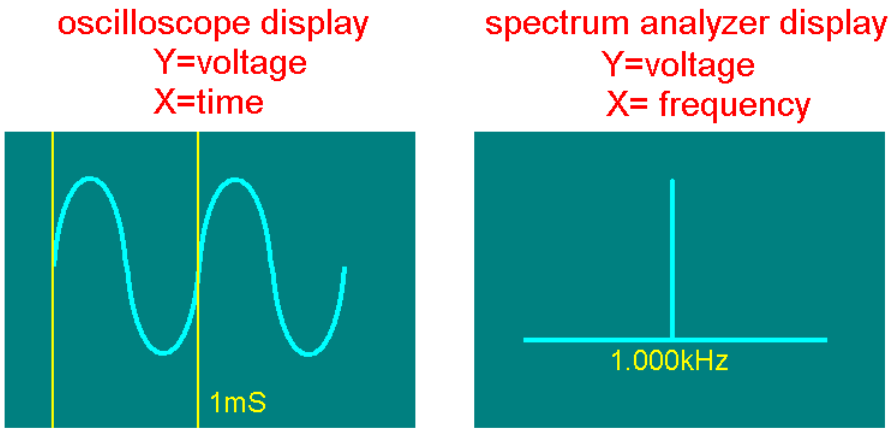
SPECTRUM ANALYZER

One of the nicest things about digital sampling is that it makes spectrum analysis easy. Spectrum analysis used to be expensive and time consuming, used in only the most advanced labs. But with digital sampling, basic spectrum analysis has become so easy, and as a result so widely available, that using it is a basic part of every tech's skill set.

A spectrum analyzer presents an X-Y graph, displayed on an LCD screen and often available for printout. In spectrum analysis, the X-axis does not correspond to time but to frequency, with lower frequencies to the left and higher frequencies to the right. Every point along the X-axis corresponds to a specific frequency.

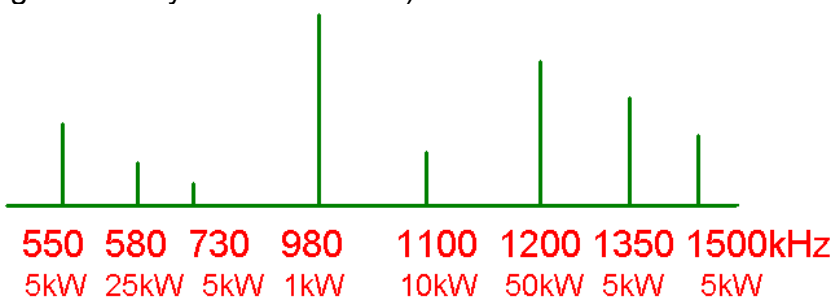
The presence of a signal at a particular frequency is displayed as a vertical line; the height of the line shows the signal amplitude. Since signals at various frequencies may differ in strength by many orders of magnitude, the vertical scale is usually logarithmic and calibrated in decibels.

The horizontal scale may display frequencies logarithmically, if it's displaying a wide frequency range, or linearly if it's displaying a narrow part of the whole spectrum. For example, in the FM radio band, the highest and lowest frequencies are about 20% apart, so a linear X-axis is best.

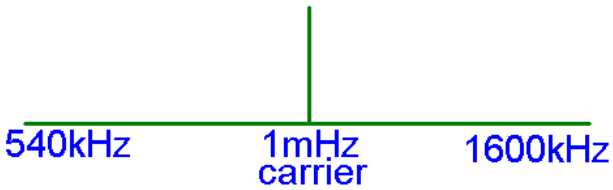


On the other hand, in the AM band, the highest frequency is more than 3X the lowest frequency, so you'd probably use a logarithmic X-axis.

Let's say you wanted to see all the AM stations on the air in a particular area. First, you would connect an AM antenna to the spectrum analyzer's input. Then you'd set the left end of its X-axis to correspond to a frequency a bit lower than 540kHz, the lowest AM channel. You'd set the right end of the X-axis to a frequency a bit above 1600kHz, the highest AM channel. Between those two extremes, the analyzer would show you the frequency and signal strength (at the receiving antenna, not the transmitting antenna) of each station as shown below. Note that, in the example, the strongest signal corresponds to the lowest-power station. This would happen if the low-power station were much closer to your receiving antenna than the other stations. (The example below does not necessarily correspond to the signals at any actual location.)

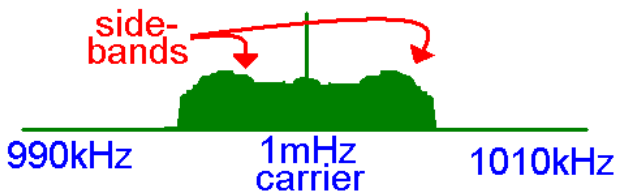


Let's say there was a station at 1MHz, on the air but transmitting silence. The spectrum analyzer would show a vertical line at 1MHz. In RF terminology, this would be called the station's **carrier frequency**.



If the station began transmitting sound, additional energy would be generated above and below 1MHz; that's how radio works as we'll see in EM6. In RF terminology, these are called **sidebands**. In theory, the spectrum analyzer should show sideband energy. In practice, all the sideband frequencies in an AM-band signal are within 0.5% of the carrier frequency, which is not far enough away to broaden the visible line on the analyzer.

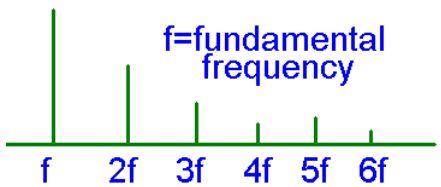
Suppose, however, that we set the left X-axis to 990kHz and the right X-axis to 1,010kHz. Now the left and right edges of the display correspond to the carrier frequencies of the next lower and next higher AM channels. The bandwidth of the station we're looking at will take up 50% of the display's width, so sidebands will be clearly displayed.



HARMONIC ANALYSIS

A spectrum analyzer can do more than display the comparative strengths of different signals. It can also analyze the harmonic and noise content of a single signal.

Suppose we feed a 1MHz signal into the analyzer and we see the display at right, with a strong signal at 1MHz and weaker signals at 2Mhz, 3MHz and so on. This means the 1MHz signal has harmonics, so it isn't a clean sine wave. If that signal is an AM radio station transmitting at 1MHz, those harmonics are illegal. The spectrum analyzer makes it possible to quickly sort out the harmonic content of a signal and alert the station's engineers to a problem. An experienced tech can even get a good idea what has gone wrong by examining the particular harmonics present.

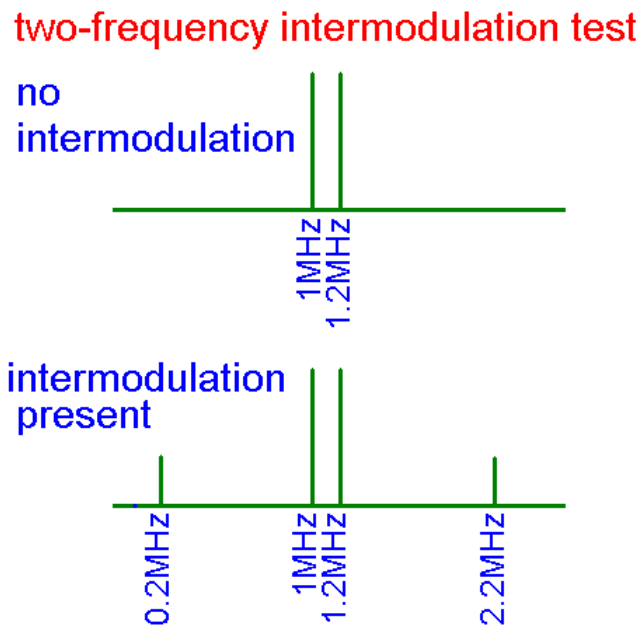


DISTORTION ANALYSIS

One of the ways harmonic analysis is used is to analyze circuit nonlinearity. The concept is simple: start with a clean sine wave. Send it through the unit under test. Then look for harmonics in the output. Since a sine wave has no harmonics except the fundamental, if the unit's output has harmonics in it, it's not putting out a sine wave. In other words, it has distorted the signal. This is called **harmonic distortion**.

Distortion can be measured on a meter, but one thing every experienced tech agrees on is that there are huge differences between the effects of one type of distortion vs. another, and a single number doesn't tell you much. Different distortion mechanisms have different harmonic signatures. Harmonic analysis can help you figure out where the distortion is originating, so you can fix it.

Another kind of distortion is **intermodulation distortion**, where two frequencies and a nonlinear circuit cause additional frequencies to be generated. A spectrum analyzer shows exactly what's going on in a way that no other tool does. In the illustration on the next page, the upper display shows signals at 1MHz and 1.2MHz only. This is what would happen if you combined two signals with a pair of resistors, so they simply summed with no measurable distortion.



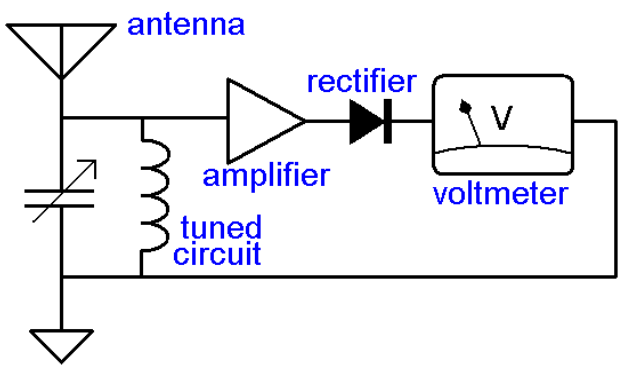
The lower display shows what would happen if you sent 1MHz and 1.2MHz signals through a nonlinear circuit. The signals would modulate each other, creating new signals at the sum and difference frequencies. The intermodulated products (the new signals) are weaker than the original ones but still significant. Whether this would cause a problem depends on the application.

SPECTRUM ANALYSIS METHODS

There are two common ways to do spectrum analysis: the **sweep method** and the **transform method**. Each has hidden pitfalls that can trip you up if you're not aware of them. In the following discussion, I'm going to show a lot of obsolete technology, because the easiest way to understand how a spectrum analyzer works is to follow its historical development.

The Sweep Method

A sweep-type spectrum analyzer uses an LC resonant tank, or nowadays its digital equivalent, which sweeps across a designated section of the spectrum looking for signals.



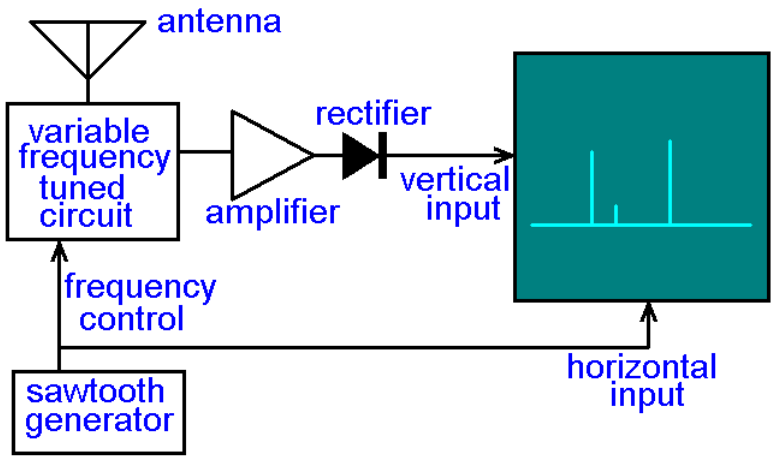
Suppose you took the tuning capacitor and inductor out of an AM radio, connected their output to a signal strength detector, and connected that output to a meter.

Suppose you tuned up and down the AM band, and whenever you came across a signal, you wrote down its frequency and amplitude. When you were finished you'd have a chart showing every signal in the AM band that was strong enough for your system to detect.

station	freq	signal (dBmV)
KXXX	580	+12
KXXY	710	-20
KXXZ	850	-9
KXYX	950	+2
KXYY	1090	-14
KXYZ	1490	-6

You could then take log graph paper and plot each of the signals you found to create a frequency vs. amplitude graph. There used to be a machine built to measure signals this way, called a "wave analyzer." This kind of spectrum analysis worked perfectly well, but it was expensive and time consuming.

Suppose you replace the knob-tuned circuit with an electronically tuned circuit using a varactor. Suppose you connect a sawtooth wave across the varactor, and then use a sawtooth oscillator to vary its capacitance, creating a tuned circuit that repeatedly sweeps the frequency band you want to look at. Suppose you took the output of the tuned circuit, rectified it, and fed it to the vertical input of an oscilloscope, and took the sawtooth wave and connected it to the horizontal input. Now you've got a machine that will automatically tune across a band, again and again, and display every signal it finds as a vertical line – an analog spectrum analyzer.

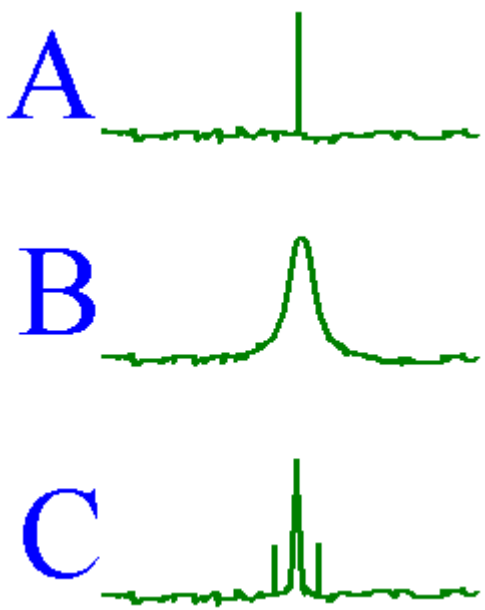


Today, these techniques are history. With digital circuitry, we don't need capacitors or inductors at all. In chapter 7, we talked about ways signals can be converted back and forth between analog and digital. We can't make a digital capacitor or inductor, but we can make a digital program that *behaves* like a capacitor or inductor – that crunches the numbers in such a way as to mimic reactive voltage, current and phase relationships – and it's easy to get the digital equivalent of a capacitor or inductor that can be varied over a 1,000:1 range, which would be difficult or impossible with actual parts. We use digital processing techniques to sweep the desired frequency band, with flexibility and precision we could never have achieved with analog circuitry.

Now for the pitfalls I warned you about.

With a sweep-type analyzer, the frequency resolution of the display, i.e. the narrowness of each vertical line, depends on how fast the analyzer sweeps across the frequency band.

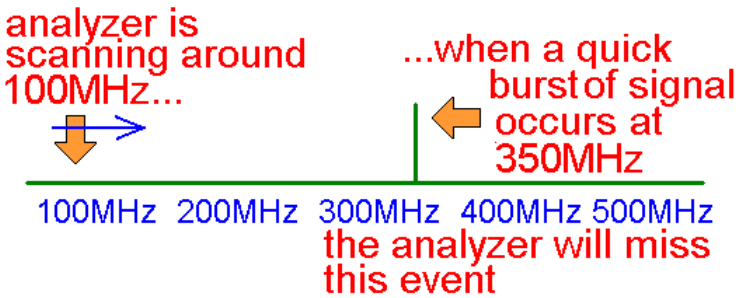
In the illustration to the right, “A” shows what the spectrum display of a strong, pure 1MHz sine wave ideally should look like. Note the rumpled bottom line. This is the **noise floor** of the system, the low-level noise that is always present, which is what a sensitive analyzer will display at all frequencies where there is no signal.



“B” shows what a display might actually look like if the analyzer is set to a fast sweep. Instead of a narrow line at the signal frequency, we see a peak at the signal frequency with slopes, called **skirts**, on either side. The skirts are a measuring artifact, they're a lie told by the analyzer. The fact that they're there doesn't mean there's really any signal at those frequencies.

Then again, maybe there is. The faster the sweep, the wider the skirts, which is to say, the poorer the resolution. If we slow the sweep way down, we may have to wait longer for the analyzer to do its thing, but we'll get better resolution as shown at “C.” And look what shows up – there were two weak signals hiding under the strong signal's skirts! One is (1MHz + 120Hz), the other is (1MHz – 120Hz). By the time you go through the EM6 study guide, you'll know that this means the 1MHz signal is being contaminated by hum from a faulty power supply.

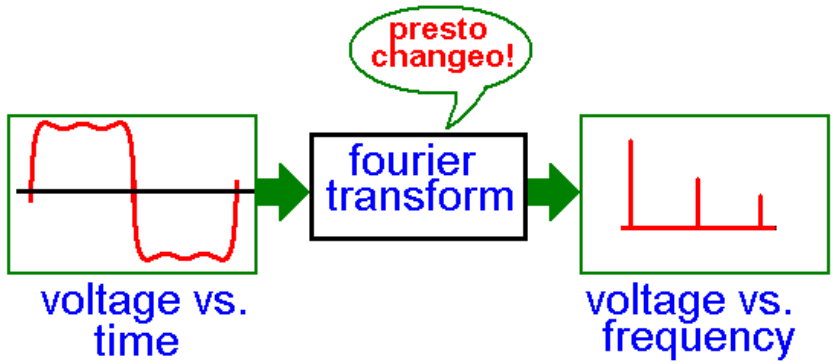
So, that's the first pitfall: sweeping so fast that frequency resolution suffers. The second potential pitfall of the sweep method is that it misses short-term events. Let's say we're scanning the band between 100MHz and 500MHz. Suppose there's a signal at 350MHz, but it only pops up occasionally and only lasts for a short time. There's very little chance that a sweep analyzer will be tuned to 350MHz at exactly the moment when the signal is present, so the probability is very high that the analyzer will miss it entirely.


analyzer is scanning around 100MHz...
...when a quick burst of signal occurs at 350MHz
100MHz 200MHz 300MHz 400MHz 500MHz
the analyzer will miss this event

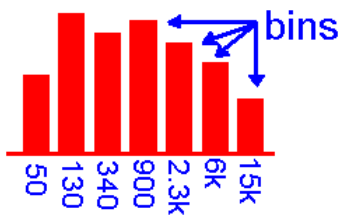
This is a serious limitation these days. It used to be that most RF signals were continuous, so the analyzer would show them, but the trend now is toward signals that occur in bursts, or are spread over a wide spectrum, or hop around among many frequencies. A sweep analyzer is not suitable for these signals.

The Fourier Transform Method

The second kind of spectrum analyzer takes the input signal as a whole and, through a bunch of complicated math named after people like Fourier and Laplace, transforms time information into frequency information. The special virtue of this kind of spectrum analyzer is that it's a **real time** analyzer; it shows what is happening right now. If a short signal burst occurs at any frequency within the band being displayed, you'll see it.



In this kind of analysis, the spectrum displayed along the X-axis is divided into **bins**, which represent narrow frequency bands. The analyzer separates out the signal energy present in each of those narrow bands and dumps that energy into the corresponding bin, like someone sorting coins into a cash drawer. You've seen audio displays that divide the audio spectrum into, say, seven bands, displaying the energy present in each band as a vertical bar graph. When I design speakers, I use a program that allows me to choose how many bins I want. Each bin represents 1/3 octave, 1/6 octave or 1/12 octave. You recall from EM3 that an octave is a 2:1 frequency ratio. At 1/12 octave, each musical tone – C, C#, D, D# etc – has its own bin, across a range of ten octaves, 120 bins in all.

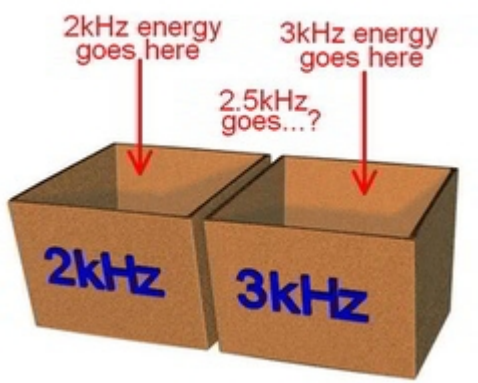


When the analyzer encounters an input signal at any frequency, it sorts that energy into whichever bin is closest in frequency.

The tricky part is that, if you take a variable frequency oscillator and slowly scan across the spectrum, there will be many frequencies that split in half and sort into two bins. Recall from chapter 7 that, when you quantize something, you lose the smooth transition between one value and the next. That's true of voltage, but it also applies to frequency. Since we've quantized the spectrum – divided it into bins – there are frequencies that don't exactly fit into any bin.

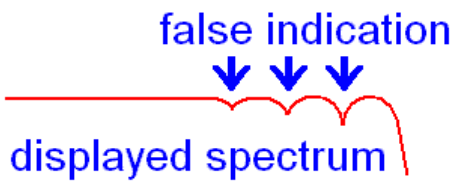
For example, imagine two bins extending from 1kHz to 2kHz and from 2kHz to 3kHz respectively. Their center frequencies are 1.5kHz and 2.5kHz. If you feed a 1.5kHz or 2.5kHz signal to the input, there's no question where it will be sorted. Suppose it has an RMS voltage of 1V; on the graph, the corresponding bin will show 1V on the Y-axis.

But suppose you feed in 2kHz? In this case, what is likely to happen is that both bins will show energy, but neither will reach as high as 1V. If you didn't know what was going on, you would guess that the signal was composed of multiple frequencies between 2kHz and 3kHz, all of less than 1V amplitude.



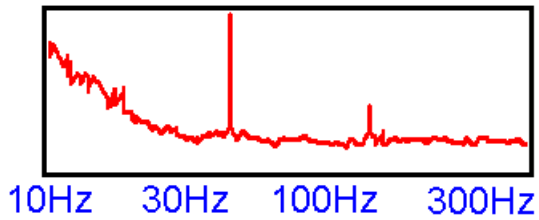
Transform-type spectrum analysis with a small number of bins isn't very good at analyzing an existing signal spectrum like, say, the AM radio band. What it's best at is analyzing how a system responds to a swept or white-noise test signal, usually supplied by the analyzer itself. For example, that's the common way of testing loudspeaker response.

On the other hand, most spectrum analyzers can easily be set to show thousands of bins. They can also include data manipulation so any input frequency will show up with its proper amplitude and X-axis location. Thus they can approach a swept analyzer for frequency resolution.



But there can still be problems. For example, if you sweep a sine wave generator across the analyzer's spectrum, at the high end – near the analyzer's Nyquist frequency – the response may resemble a comb filter. This is a measuring artifact that is created as the sine wave generator moves from one bin to the next. As always, the best techs are those who have become familiar with every characteristic of the test gear they use.

Low Frequency Noise

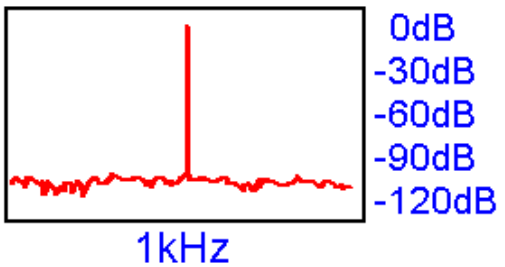


When the frequency range of a real-time analyzer is set to include very low frequencies like 20Hz, you can get a display like this one. The display shows energy at 60Hz, which could indicate a signal or power line hum, a bit of 3rd harmonic at 180Hz, and what seems like a lot of low-frequency noise.

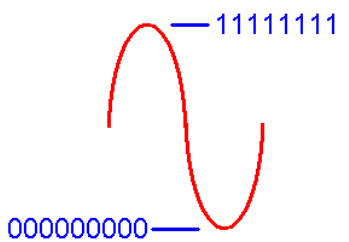
This is tricky, because if you're looking at a very low-level signal, the noise might be real. Most solid-state analog circuits exhibit a rise in the noise floor at low frequencies. But it can also be a measurement artifact; the apparent noise due to randomness associated with a small number of samples. To make accurate measurements at low frequencies, you need to spend some time. The analyzer must be set to average values of many samples over many seconds. That will remove the randomness from the measurements and the false low-frequency noise will go away.

Sampling noise gain

One of the interesting things about a spectrum analyzer is that the displayed noise floor may be much lower than the actual noise floor of the unit or system being tested. For example, the display at right represents a 1kHz tone as digitized by a 12-bit A-D converter. The 1kHz tone is at maximum level and the noise floor is around 110dB below this maximum, indicating a 110dB dynamic range.



But a 12-bit A-D does not really have a 110dB dynamic range. The largest sinewave that an A-D converter can handle is one whose positive peaks result in a digital output that's all 1s, and whose negative valleys result in a digital output that's all 0s. An A-D converter's noise floor is lower than this maximum level by about 6dB times the number of bits, plus 2dB. In the case of a 12-bit ADC, that's 12 x 6 = 72, plus 2, for a total of 74dB. That is, a 12-bit A-D converter's quantization noise is about 74dB below the maximum sine wave level.



So, how does it happen that the spectrum analyzer shows a lower noise floor? It happens because, for each individual bin, only the noise *in that bin*, the noise that exists within the small frequency span represented by that bin, is shown. In EM3, we discussed the fact that the white-noise energy existing across any span of frequencies is proportional to the square root of the span. If you double the number of bins, cutting the frequency range of each bin by half, the noise in each bin will drop by 3dB. This is what, in the context of a spectrum analyzer, is called **noise gain**. (The expression is also used to mean different things in other cases.) With 4096 bins, noise gain is over 30dB.

This is both potential pitfall and advantage. Pitfall, because it might lead you to think the noise floor is much lower than it really is, which for example might cause you to think a preamp is okay when it's actually out of spec by 30dB. Advantage, because the analyzer is able to extract and display weak signals that are far below the noise floor.

OSCILLOSCOPE

I've put off the oscilloscope discussion because modern digital oscilloscopes incorporate many of the features of logic analyzers and spectrum analyzers. If you've gone through the EM2 study guide, you know about analog scopes. However, many students do EM4 first. So, I'm going to begin with a review of analog oscilloscopes.

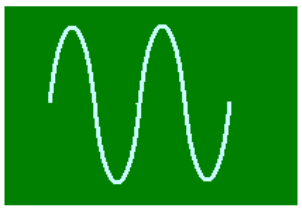
ANALOG OSCILLOSCOPE REVIEW

THOSE WHO HAVE DONE EM2 AND THE SCOPE DISCUSSION MAY SKIP AHEAD TO PAGE 36.

A lot of people service digital equipment using analog scopes. Even the best ones are inexpensive and do a good job with high frequencies.

Suppose you took a pencil and started to draw a line from left to right across a paper, moving at a constant rate. Suppose, halfway across, someone bumped your hand toward or away from you, but did not bump it sideways, so as not to disturb your steady progress across the paper. When your hand got to the right side of the paper, you'd have a visual record of the event.

That's what an oscilloscope does. In an analog scope, there's a CRT (cathode ray tube) with a phosphor on the inside front face. An electron beam scans across it from left to right, energizing the phosphor and leaving a line called a **trace**.



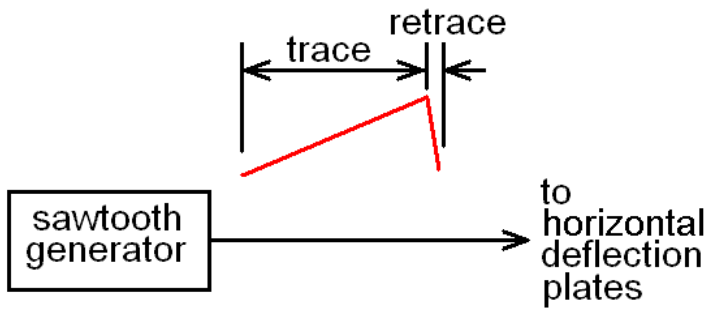
If there is no input signal, the beam simply draws a straight line. But if there is an input signal, it causes the electron beam to be deflected up and down, resulting in a visual display of the waveform. In other words, an oscilloscope draws a real-time, electronic X-Y graph of the signal waveform.

RETRACE

In EM2, when I showed a repeating waveform, I showed a single cycle. In a diagram, that's all that needs to be shown, since every wave is the same as every other. However, when we look at a waveform on an oscilloscope, we're not looking at a theoretical representation, but a signal occurring in real time. If it changes, we want to know. So the scope has to continually display what's going on.

Suppose we want the beam to move fast enough so a single cycle of 1kHz sine wave appears half an inch wide on the CRT face. This means the beam must move half an inch per millisecond horizontally, or 500 inches per second. If we want to watch a minute's worth of that, we need a screen that's 2,500 feet wide - upwards of half a mile.

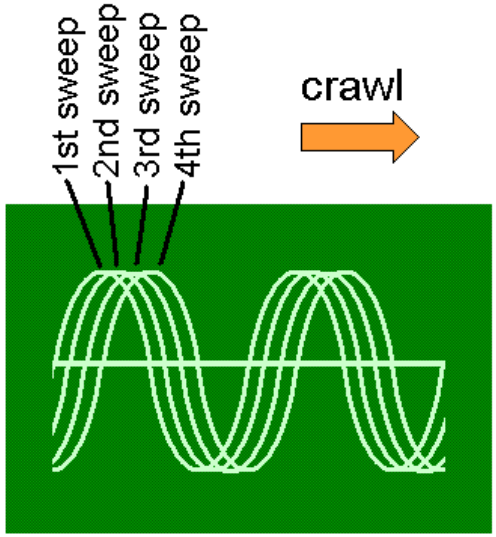
How do we fix that problem? We build the scope so every time the beam reaches the right side of the screen, it snaps



back to the left and repeats the same left-to-right motion, over and over again. If the signal waveform hasn't changed, and the scope's scanning frequency is the same as the signal frequency, the beam will trace the same path again. When the signal waveform does change, we'll see it.

In an analog scope, this is done by a **sweep generator** which creates a **sawtooth wave** – one that rises slowly and falls quickly – used to make the electron beam sweep across the display from left to right. During the slow rise, the beam draws a picture of the input signal waveform. During the rapid fall, which causes the electron beam to snap back to the left, the electron beam is turned off or **blanked** so as not to create a confusing pattern. This short time period is called the **retrace interval**.

SYNCHRONIZATION



The biggest remaining problem is synchronizing the sweep frequency – the frequency at which the trace repeats – with the incoming signal frequency. Suppose we want to look at a 100Hz sinewave, but the sweep generator is running at 97Hz. That means the second sweep will start late, when the sinewave is already 3% through its cycle. On the third sweep, the wave will start 6% ahead of the sweep, on the fourth sweep 9% ahead and so on. This will shift the position of the displayed wave a little with each sweep, making it crawl across the screen.

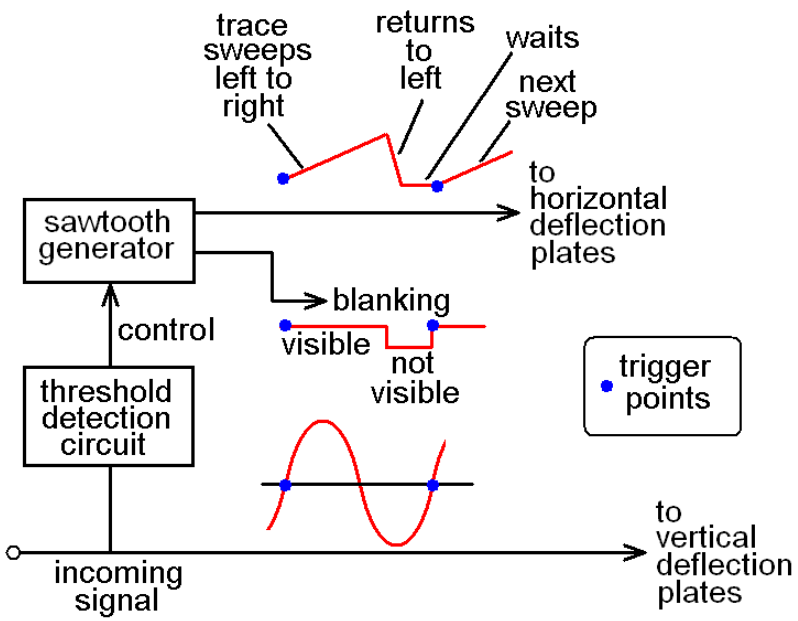
The displayed signal waveform will only stand still on the screen if the sawtooth generator is running at precisely the same frequency as the signal. At high frequencies, say 10MHz, it's almost impossible to manually adjust the sweep frequency precisely enough to get a stable display.

We solve this problem by adding a threshold circuit that monitors the instantaneous voltage of the input waveform, looking for the moment when it crosses the zero-volt line, going from negative to positive (blue dots in the diagram tot he right). When the beam completes a sweep, it snaps back to the left, but waits for the threshold detection circuit to tell it to start the next sweep.

This is called sweep **triggering**, and refers to the fact that each sweep is triggered when the signal crosses the zero line going in the positive direction. If the waveform doesn't change from one sweep to the next, the trigger will hit at the same point in each wave and the electron beam will keep tracing the same wave shape in the same position, resulting in a stable display.

This eliminates the need to synchronize the sawtooth sweep generator with the signal frequency. The electron beam traces the signal waveform for as long as it takes to cross the screen, then it returns to the left and waits for the trigger circuit to tell it to start the next sweep.

THE TRIGGERED SWEEP



You, the user can set the sweep to whatever speed you like. This is an especially useful feature because many real-world signals change in frequency or are a complex mix of frequencies, and couldn't be synchronized any other way.

Adjustable Triggering

We may not always want triggering to occur at the point where the signal waveform crosses zero. For one thing, some complex waveforms cross zero more than once each cycle.

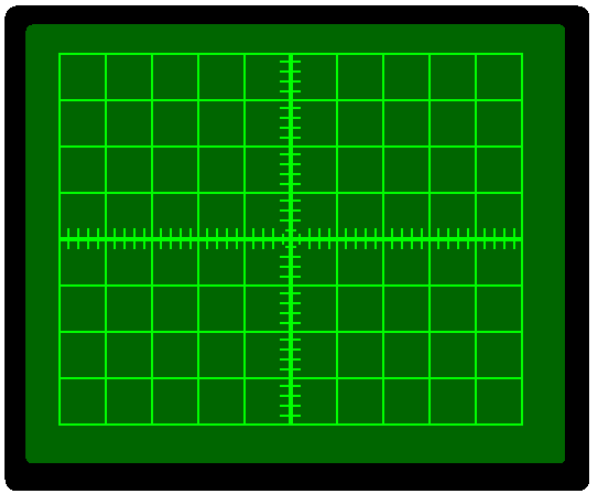
An **adjustable trigger** allows us to set the threshold voltage. Most scopes also allow us to flip the trigger polarity over, so the sweep triggers when the waveform crosses the threshold going in the negative direction.

Automatic Triggering

Most scopes offer the option of **automatic triggering**. This is a circuit that monitors the input voltage and automatically sets the threshold so that, no matter how large or small the signal, there will always be a point where the sweep is triggered. This is handy because, if the threshold does not happen to be set to zero, a small signal may not trigger a scan at all, and the screen goes dark. Not a crisis, but it is an annoyance to have the trace disappear when the signal voltage is low.

Most techs leave the triggering set on automatic most of the time, and only turn to manually-adjusted triggering when they're dealing with a funky waveform that the automatic circuit doesn't know how to handle. Or, they may choose manual triggering when they want to closely examine the waveform right where it crosses the zero line, so they need the sweep to be already underway by that point.

SCALE GRADUATIONS

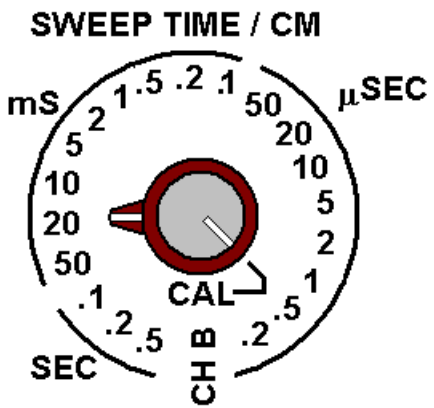


The oscilloscope has horizontal and vertical lines on the face which form a grid of squares. This illustration shows a scope with 10 major graduations in the horizontal direction and 8 vertical ones. The X- and Y-axes have 5 minor graduations per major one. The major graduations are typically one square centimeter each, a common dimension for a full-sized bench scope.

If we know how fast the trace sweeps across the screen, the lines can be used as reference points to visually determine the period of the signal waveform. A quick mental calculation if you're macho, a few pokes at the calculator if you're not, gives you the frequency. Likewise, the gradations can be used to determine the signal voltage at any point in the wave. I'll show you how in a bit.

HORIZONTAL RANGE SWITCHING

A trace ought to be able to move so slowly that it can display signal voltage changes that take several seconds to happen, or so fast that it can show the highest frequencies the scope is capable of handling. In a typical scope, the slowest sweep takes five seconds to go across the screen; the fastest sweep occurs in a couple of microseconds.

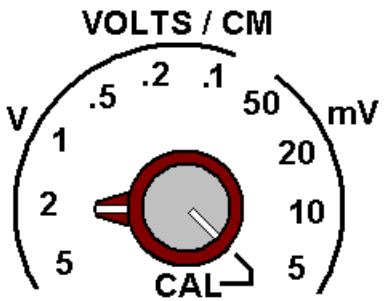


Sweep speed is typically selected by a rotary switch calibrated in increments of 1, 2, 5, 10, 20, 50, 100, 200, 500 and so on. The slowest speed might be half a second per major division on the scope face. If the scope face has 10 divisions, the trace would take 5 seconds to sweep. The fastest speed might be 10^{-7} second per division, meaning the trace would complete a sweep of 10 divisions in 10^{-6} seconds.

At 10^{-7} seconds per division, a 10MHz sinewave would display as one cycle per division, ten cycles across the screen.

Chapter 10 page 32

VERTICAL RANGE SWITCHING

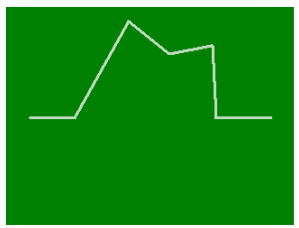


Likewise, a scope ought to be able to display waveform voltages from small to large. Waveform height on the screen is controlled by routing the signal through an attenuator / multiplier resistor arrangement like the one in a multimeter, which we studied in EM1. The signal-attenuating switch is calibrated in the same increments as the sweep switch: 1, 2, 5, 10, 20, 50 and so on. When a typical scope is set to maximum sensitivity, a 5mV peak-to-peak signal will occupy one vertical major division. At minimum sensitivity, a 5V P-P signal would occupy the same space.

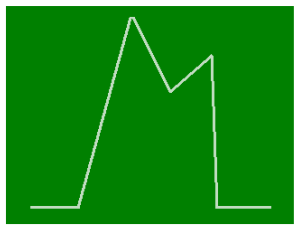
VERTICAL AND HORIZONTAL POSITION

You will often want to move the whole trace up or down or to one side. To begin with, the trace doesn't automatically position itself halfway up the screen, you have to put it there. You'll want to carefully place it right on top of the center line, so when the signal moves it up or down, the displacement can be measured against the division and subdivision lines.

Then again, you may choose to put the trace somewhere else. If you're looking at a waveform that never goes negative, you'll be able to see it more clearly if you move the trace down to one of the lower division lines, then change the vertical range switch to fill the whole screen with the waveform.



centered trace



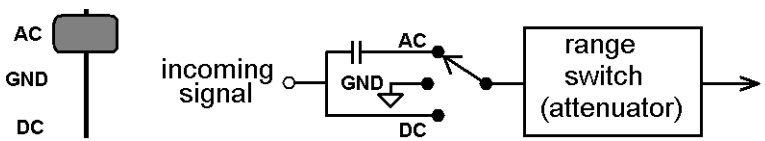
shifted trace

The circuitry inside a piece of equipment is typically powered by a +DC rail. If you set the trace low on the screen, corresponding to ground which is the lowest voltage in the circuit, and +DC high on the screen, corresponding to the highest voltage in the circuit, every voltage at every point in the circuit can be displayed on the screen.

You may also want to move the horizontal position. This usually results in part of the trace going off the screen to right or left, so you won't be leaving it that way. You move it just long enough to line up part of the waveform with one of the vertical lines, so you can count divisions and calculate period or frequency.

It is often necessary to examine a small AC voltage riding on a large DC voltage, e.g. when you're following a signal through a defective amplifier to find the point where it's getting lost. The DC voltage will move the trace up or down. If you try to set the range switch so you can see the small AC component, the DC component will push the trace off the screen.

This is the reason for the **DC/AC/GROUND switch**. When set to the DC position, the switch passes the signal straight through to the range switch. When set to AC, it switches in a series capacitor.

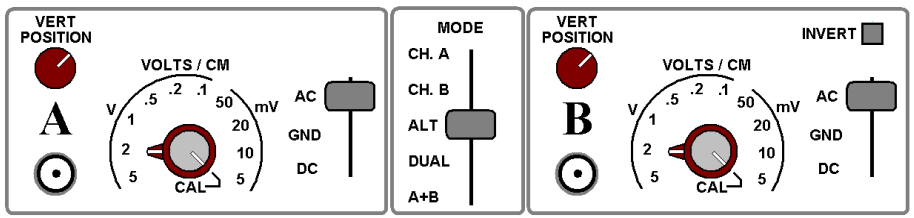


The capacitor passes the AC part of the signal but removes the DC component, allowing you to turn the vertical sensitivity up as much as you like. The "ground" position disconnects the signal and connects the range switch to ground. This is helpful in quickly adjusting the vertical position knob to place the trace where you want it.

DUAL TRACE SCOPES

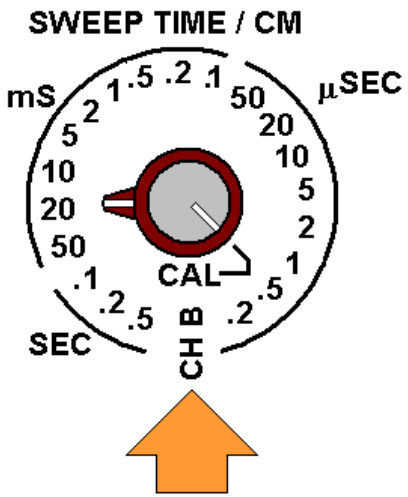
All but the most inexpensive scopes can display two separate traces. This is useful in comparing, say, input and output waveforms to see what a piece of equipment is doing to a signal.

The sweep rates are not independently adjustable; the whole point is to compare two waveforms occurring in the same time frame. However, the input attenuators are independently adjustable, so you can simultaneously display a small signal and a large one.



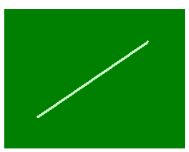
The controls for each input, A and B, are self-explanatory; each trace is provided with the same controls as the trace on a single-trace scope. Each has a vertical position knob, so each trace can be moved up or down on the screen. They can even be superimposed for close comparison.

THE X-Y SETTING

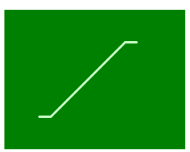


There's one position on the sweep speed switch that has nothing to do with sweep speed. In the **channel B** setting, the beam doesn't sweep at all. Instead, the B-channel signal is used to deflect the beam horizontally, while the A-channel signal deflects it vertically. In the absence of signal, the beam sits in the center, making a glowing dot. (Turn down the intensity when using this setting.) This turns the scope into a graph-drawing machine that displays A as a function of B. Let me give a few examples to show how useful this can be.

In figure 1 to the right, the same signal is connected to A and B inputs. The display is a line tilted upwards at 45 degrees, as you'd expect; for every CM of horizontal movement, there's an equal CM of vertical movement.

1 

In figure 2, the same signal as before is connected to input B, but the signal goes through an amplifier on its way to A. The flat top and bottom shows that the amp is beginning to clip (overload).

2 

In figure 3, the amp is beginning to overload, but the distortion comes on gradually. Looking at a sine wave, you'd never see this; gentle distortion of a sine wave must reach about 10% before it is obvious.

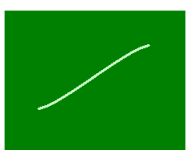
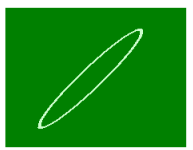
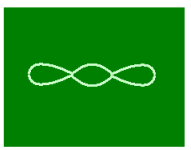
3 

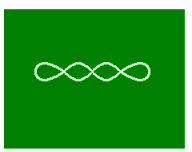
Figure 4 shows a phase difference between the signal at A and at B. If you were adjusting the relative phase of two sine waves, you could feed one to the A input and one to the B input, then vary the phase of one of the signals until the loop closed up. This works best with high-quality scopes; inexpensive scopes often have some built-in phase difference between horizontal and vertical circuitry, thus can't give you a reliable phase indication at high frequencies.

4 

5

Figures 5, 6 and 7 show that the frequency of the sine wave at A is twice, three and four times the frequency of the sine wave at B. These are called **Lissajous patterns**.

6 

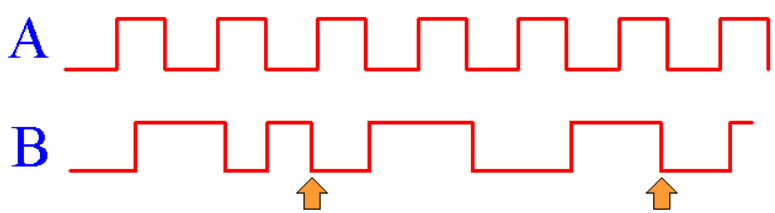
7 

A single-trace scope usually has a separate horizontal input for the X-Y setting. This isn't quite as good, but it works. The advantage of a dual trace scope is that the X-axis deflection has the same range of control as the Y-axis, with the same calibration.

THOSE WHO SKIPPED AHEAD, PICK UP HERE.

DUAL TRACE SCOPES AND TIMING DIAGRAMS

In digital applications, the most important use of a scope's dual trace feature is to compare the timing of two data streams. Suppose, as in the illustration below, input A shows a continuing clock, and B shows a data stream with jitter. Note that there are two points where a data transition comes before the corresponding clock. Unless the unit under test has circuitry to detect and correct this, it's a problem.



The usefulness of a dual trace comes, in large part, from the fact that the triggering and sweep rate are the same for both traces, so direct A-to-B timing comparisons can be made. In chapter 4, we talked about the importance of timing diagrams and proper sequencing. The dual trace function can be used to verify that the timing sequence in a circuit is (or is not) correct.

In a few paragraphs, we will be talking about the attenuator switch found in many scope probes. When you're looking at timing of data streams, you want the probe set to 10X to keep the scope from loading down the data stream. Otherwise, the capacitance of the probe lead might cause timing errors that do not exist in normal circuit operation.

THE DIGITAL OSCILLOSCOPE

The basic display function of a digital scope is exactly the same as an analog one. The X axis represents time, the Y axis represents voltage. Instead of an electron beam physically sweeping across the screen, a digital scope takes many samples during a specified time interval, placing the samples in memory. Then, when it has one scan's worth of samples, it places a reconstructed waveform on the display.

There are a number of advantages to the digital method.

- In an analog scope, it is difficult to provide more than one electron beam. If we want two or more traces, the beam has to be shared back and forth between them, which creates some problems. In a digital scope there is no built-in limitation to the number of traces; that's a design choice by the manufacturer.
- The displayed waveform can be captured and stored as long as desired. There are analog scopes that provide a storage function, but it's very primitive. Digital scopes can store a large number of events, and can be connected to a computer to store even longer sequences.
- A long sequence of scans can be called up later for examination, limited only by the scope's memory.
- The scope can measure various aspects of the signal, e.g. peak voltage, and show the numbers on the screen along with the waveform. Some analog scopes can do this too, but it takes very complicated circuitry. With a digital scope the data is already in memory, so it's simple for the scope to analyze it.
- The scope can transform the signal waveform in various ways, e.g. change from a normal voltage-vs.-time display to a voltage-vs.-frequency spectrum analysis via Fourier transformation.
- A dual-trace analog scope can usually show the sum or difference between the waveforms at its A and B inputs. A digital scope can usually show A x B and A/B as well.
- Without a CRT, which is typically much deeper than it is wide or high, the scope can be light, compact and resistant to impact damage.

The scope's X and Y scaling controls (time per division, voltage per division) work the same way as with an analog scope, except that a digital scope can do the scaling automatically if you want.

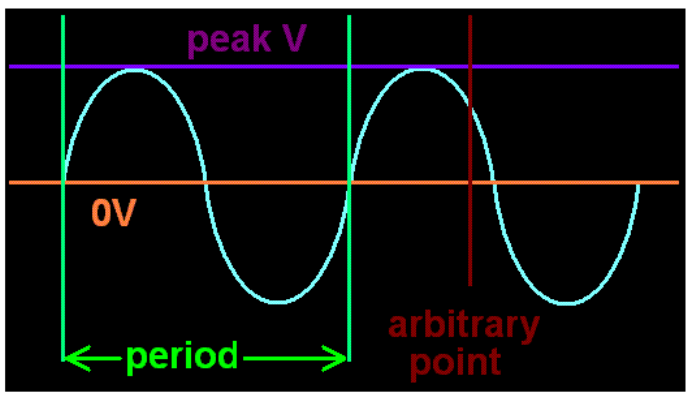
A digital scope almost always has two input channels, often four or more, blurring the distinction between a scope and logic analyzer. All except the most basic scopes perform automatic measurement functions, displaying the results on the screen, which with an analog scope you would have to make by counting scale divisions. Typical measurements:

- peak voltage and Vpp
- average voltage
- RMS voltage
- voltage at any point in the waveform, which you select by moving a **cursor**
- frequency and period
- highest and lowest voltage points
- digital 1 and digital 0 voltages
- overshoot, risetime, falltime, duty cycle
- delay between risetime at one input and risetime at another, i.e. propagation delay

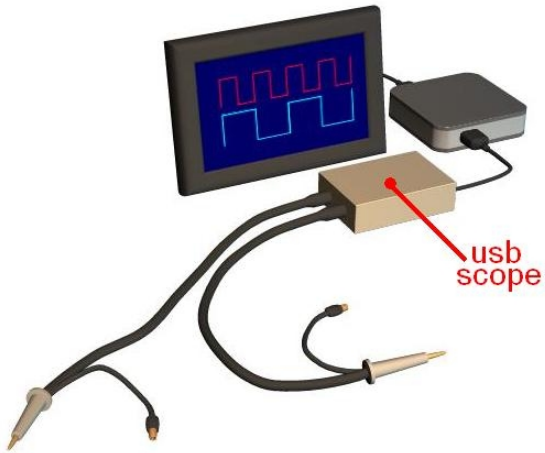
Many digital scope displays do not have the grid of squares that an analog scope would have, because there is no need for the user to count divisions – the scope makes those measurements for you.

CURSORS

I mentioned cursors, which are vertical and horizontal lines that are positioned at points in the displayed waveform. In some cases, the scope positions them. This gives the user a visual indication of what is being measured. In other cases, the user positions them, permitting measurement of voltage at any arbitrary point in the waveform.



USB SCOPES



Some scopes blur the distinction between scope and computer, too: they take the form of small boxes with probe connections on one end and a USB port on the other that you plug into your computer. I should mention that regular digital scopes have computer connectivity too, so you can save and process waveforms. Even the least expensive digital scopes can store and retrieve waveforms, but their memory is limited.

HIGH FREQUENCY LIMITATIONS

Like an analog scope, a digital scope's response begins to fall off at some high frequency. To avoid the problems that occur when the input signal is near the Nyquist frequency, which we talked about in chapter 7, this rolloff begins at ¼ to $^1/_{10}$ the sampling frequency. A low-priced scope is usually good to 25MHz; expect to pay $500-$1,000 for one that goes to 100MHz.

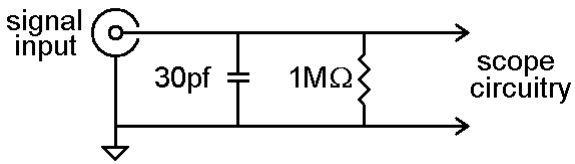
With all the digital scope's advantages, is there any reason ever to buy an analog scope? There may be.

- Used basic analog scopes are really cheap. If you want one of your own and you're living on a student's budget, you can get something useful for $100-$200. You can't get much in a digital scope for that amount. But keep your eyes open; digital scope prices keep coming down.
- A digital scope that can display really high frequencies is very expensive.
- A digital scope, as you'll find when you start using one, has quirks related to sampling and quantizing that an analog scope doesn't have. With inexpensive scopes especially, there is a delay between when a signal comes into the scope and when you see a stable display of it – while analog scopes are real-time.

Just during the time period when I've been writing these study guides, technology has advanced so much that I often had to go back and change the text, even before publication. So, expect the analog scope's remaining advantages to disappear over the next few years.

INPUT LOADING

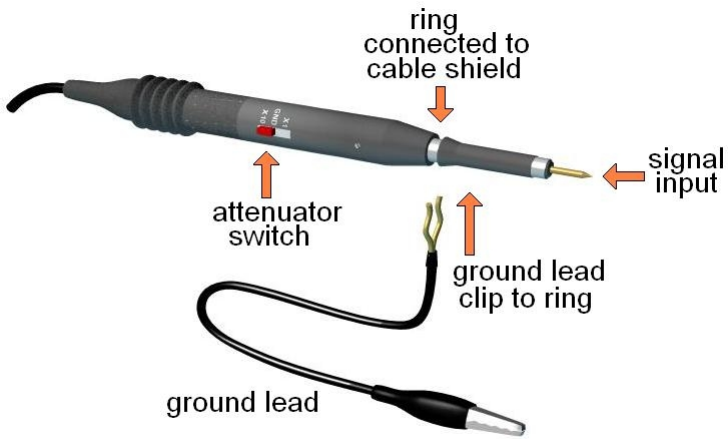
As with a VOM, the user must always keep in mind that the act of attaching a measuring device may upset the measurement. The inputs to an oscilloscope usually have about a 1 megohm resistance in parallel with as much as 30pf of capacitance. The resistance is fairly high, but capacitive reactance falls below 1MΩ already at 5kHz. When the signal frequency reaches 5MHz, this input is essentially a 1kΩ capacitive load, which is a pretty heavy load to be attaching to the innards of a circuit. Moreover, measurements are taken via probes attached to the inputs, and that adds more capacitance. A typical probe connected to the scope via a 3-foot cable adds 100pf. Better probes, using special low-cap cable, have less capacitance.



Bottom line: simple probes aren't good for measuring frequencies over a few hundred kHz, unless you're looking at a point in the circuit with very low impedance.

ATTENUATOR PROBES

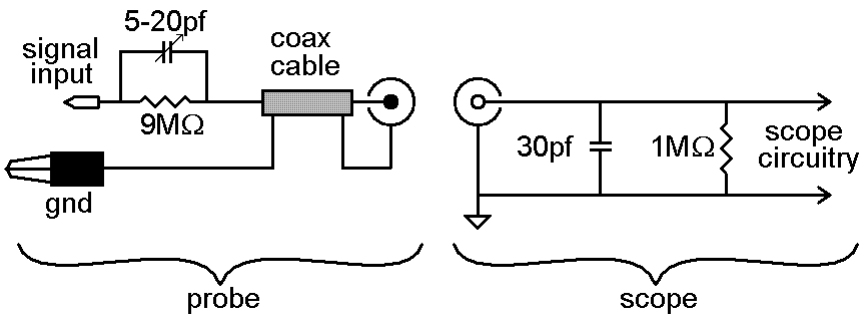
An attenuator probe has a switch that allows you to insert a 10:1 voltage attenuator, right in the probe. This has two uses: reducing the load on the circuit you're measuring, and increasing the maximum voltage the scope can handle.



If we go back to the input attenuator of our representative scope, we see that the least sensitive setting is 5V/CM. When the switch is rotated to this setting, a 5V change at the input will move the trace one major division up or down. Since there are eight major vertical divisions, this means the largest waveform that can be shown is 5 x 8 = 40 volts P-P. This can be increased by rotating the "fine" knob, but then we lose the calibration.

Part of the reason scopes aren't made to handle greater input voltages is that it isn't a good idea for those voltages to appear on the input connector, which is usually a "BNC" type that's an excellent connector from a signal-handling standpoint, but isn't designed for high voltages.

The proper way to display a high voltage is to switch in the 10:1 attenuator in the probe. This inserts a 9MΩ multiplier resistor and small capacitor in series with the input, creating a voltage divider that knocks the signal down before it even gets to the shielded cable. This converts the 5V/CM range switch setting to 50V/CM, allowing the scope to display a signal up to 400V P-P. Using a scope this way requires some presence of mind on the part of the user; you have to remember which position the probe switch is in, or your measurements will seem to be off by a factor of 10:1.



The other thing that happens when you switch in the 10:1 attenuator is that the input impedance goes way up, which is to say the loading goes way down. In the 10:1 setting, the multiplier resistor is paralleled by a small variable capacitor that forms a capacitive voltage divider with the input and cable. Typically, the final capacitance is about 12pf. Deluxe probes have less.

I very strongly recommend leaving the probe in the 10X position, except when measuring very small voltages. With an analog scope, you would have to mentally multiply the input attenuator setting by 10. With a digital scope, it's usually possible to tell the scope to shift everything by 10X, so the numbers you read on the scope face are the actual voltages.

THE GROUND LEAD

I've mentioned this before, and I'll talk a lot more about it in chapter 10. It's not good enough to just get the scope input connected to the unit you're testing. You also have to get the scope ground connected to the unit's ground. And, as you'll see in chapter 10, "ground" is a slippery concept.

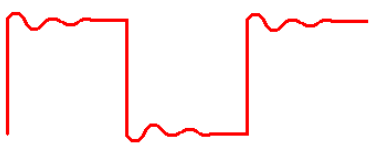
A test probe has a ring near the tip that's connected to the coax cable shield. The probe comes with a short lead that attaches to this ring, with an alligator clip at the other end to attach to ground.

This is the ground lead you're supposed to use to attach to the chassis of the unit you're working on.

You're going to break that rule a million times. The ground lead is often too short to allow the probe to reach the point in the circuit where it has to reach. You'll find yourself using another clip lead to extend the ground, and many times, in frustration, you'll just run a “wild” ground wire from the scope itself – there's almost always a ground terminal somewhere – to the unit under test. Problem fixed.

But there's a reason the supplied ground lead is so short, and you should know what it is so you'll know how bypassing it might affect your measurements.

At low frequencies, one end of a grounded wire is the same as the other end. But at high frequencies, there's a time difference between the ends of the wires. this means, at 100MHz, one end of a 4-foot test lead isn't firmly connected to the other end. It isn't so much a cable as a tunnel, a conveyance that guides waves along a path. For this to work, the ground – no, the *reference* must travel right along with the signal. If the reference goes from unit under test to scope by any other path, things can get screwed up.

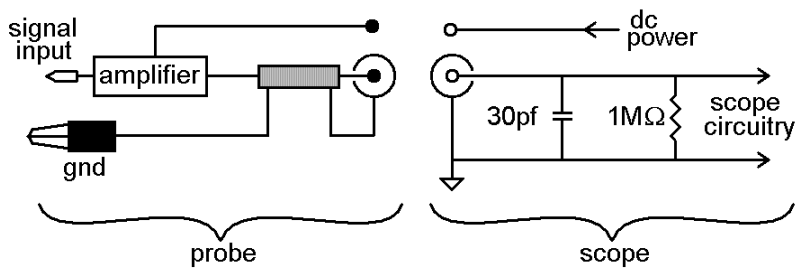


For example, in a square wave, you can get **ringing**, which is related to the resonant behavior of a bell. The waveform you're looking at may actually be ringing, or it may be an artifact caused by careless grounding. It's important to know which it is.

I'm not saying you should never bypass the “system." I myself am extremely careless with scope grounding, but then, I usually work with frequencies under 50kHz. At 100MHz, and especially with digital signals which are not sinewaves, the waveform on the scope will bear no resemblance to what's going on in the circuit unless you treat the test lead as a transmission system, not just a wire. If you're testing something, and you haven't been careful with the grounds, and the waveforms seem to be full of strange glitches, clean up the grounds and look again.

ACTIVE PROBES

At frequencies of hundreds of megahertz, even the small capacitive load of a 10X probe is a problem. High-speed scopes isolate the test points from the cable capacitance by building amplifiers right into the test probes. The amplifier, not the input signal, supplies the current necessary to drive the cable and scope input.



An active probe can have a capacitance as low as 1pf. You have to be careful with it; if you break one, you'll have to get another identical one from the scope manufacturer. To my knowledge, nobody makes a generic amplified probe that will work with a wide range of scopes. And active probes are not cheap.

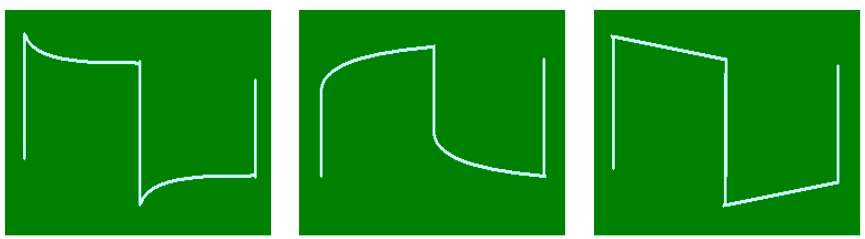
The easiest way to damage an active probe is to connect it to a voltage that's higher than what it's designed to handle. Remember, the scope's range switch comes after the probe, so turning it down doesn't reduce the voltage that the probe's amplifier sees. Read the owner's manual so you know its limits. As a rule of thumb, most active probes cannot tolerate an input voltage greater than 15V.

This is one case where static prevention – grounded bench, grounded soldering iron, grounded unit under test, grounded technician – is very important. Even if there's no voltage in the unit under test that exceeds 15V, static can easily be a hundred times higher.

CALIBRATION AND SCOPE LIMITS

The biggest mistake people make when using test gear is to assume what they see is automatically true. It can't be said often enough, and though it applies with special force to oscilloscopes, it also applies to all other test gear: you need to become so intimate with it that if you cheat on your gf/bf it'll know. Wizard-level techs even study the internal circuitry. You'd be surprised how often that comes in handy.

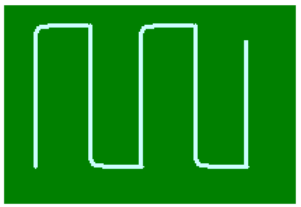
A scope is not really ready for use until you've adjusted the probes. There's a little screw head in the side of the probe to make the adjustment, and most scopes have a square-wave output terminal on the front to use for calibration. A square wave should have a flat top and bottom; if it doesn't, voltage measurements will be off.



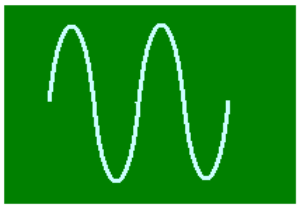
excessive high frequency response | deficient high frequency response | deficient low frequency response

Fortunately, with digital scopes, the scope itself rarely requires tweaking.

You need to become familiar with a particular scope's limits. For example, if your scope won't display anything over 15 MHz, a 10MHz square wave is going to look like a sine wave. That square wave's lowest harmonic is 30MHz (square waves don't have even numbered harmonics), so the fundamental is the only thing that will get through.



10MHz square wave on a 50MHz scope



10MHz square wave on a 15MHz scope

You also need to get a good feel for apparent anomalies that are natural results of the measurement process. Suppose you're looking at a low frequency square wave with the input switch set to AC, and you see some tilt. That's what happens when you put a capacitor in series, it's normal.

Does it look like there's some DC on the input, when there isn't supposed to be? Switch it to GND and check to be sure you haven't accidentally bumped the vert position knob.

Age isn't the only thing that makes techs go bald. They also pull their hair out, unable to locate a nagging problem in a unit they're servicing, because the problem isn't in that unit, it's in the way they're using the test gear. So now you've been alerted.

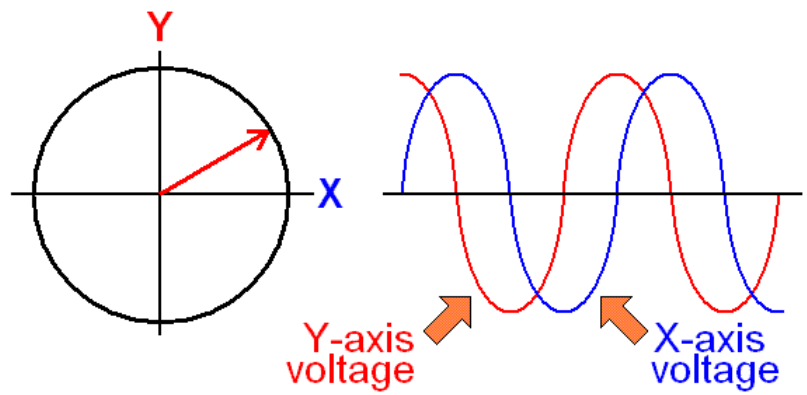
We were just talking about how a scope probe loads a circuit. The only thing you skipped over that's new is that, with the high frequencies and not-so-low impedances in digital circuitry, you *always* set the probe to 10X attenuation unless you're measuring a very weak signal. The capacitive loading of a 1X probe is so heavy, at modern data frequencies, a CMOS output cannot drive it.

VECTOR ANALYZER

Spectrum analyzers, and scopes with an FFT function, display signal strength vs. frequency. Scopes can also be set up to display phase. A **vector analyzer** displays phase naturally, as one of its built-in functions.

This is important because one of the ways of transmitting more information in a given bandwidth is to vary both the voltage and the phase of the signals. I'm not going to go into much detail about this here, because it will be more fully covered in EM6. But you need to know a little about it to understand vector analysis.

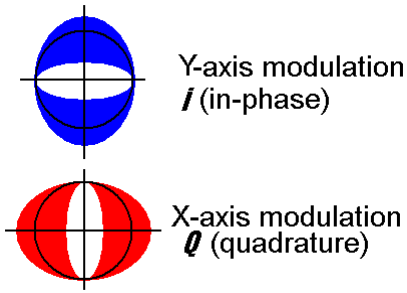
Let's go back to EM2 and the discussion of waveforms. Recall that a sine wave is a phasor's Y-axis position vs. time. Now, the phasor is actually moving in the X-axis just as much as in the Y-axis. If we plot X-axis and Y-axis movement together, we'll get two sine waves, 90^{O} out of phase with each other.



Recall that **modulation** means varying some aspect of a signal. If we decide we're going to vary the amplitude, we can vary the Y-axis only or the X-axis only.

Suppose we take a blue pen and draw circles around the origin, the point where the X and Y axes cross. Suppose we vary the distance between the origin and the pen in the Y-axis direction, but we always keep the same distance on the X-axis.

After many passes around the origin, we will have the blue pattern to the right If our blue pen represented the tip of a phasor, we would be varying the phasor's length when it's pointing up or down but keeping it constant when it's pointing right or left. On the other hand, if we take a red pen and made another image, keeping the phasor length constant when pointing up or down but varying it when pointing right or left, we'll get the red pattern.

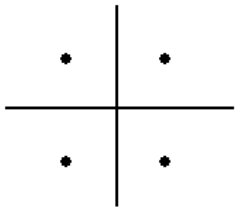


The important thing here is that X-axis modulation and Y-axis modulation can be completely independent. We can modulate X and Y with two different signals. We've just doubled the amount of information that can be encoded in a transmission. Cool-o-matic.

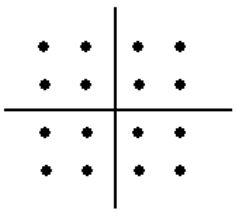
Terms: modulation that causes the length to vary in the Y-dimension is called **in-phase** modulation, identified by a letter **I**. Modulation at 90 degrees away from Y – modulation of X – is called **quadrature** modulation, identified by a **Q**.

If we carefully adjust I and Q modulation, we can control the sine wave's peak amplitude and phase. In other words, we can place its peak at any point on an X-Y-graph.

We can take that idea further. Suppose we place four dots, one in each quadrant, and make them represent 00, 01, 10, and 11 respectively. By suitable adjustments of the I and Q modulation, we can hit any of these points, as if we were throwing darts.



4-point constellation



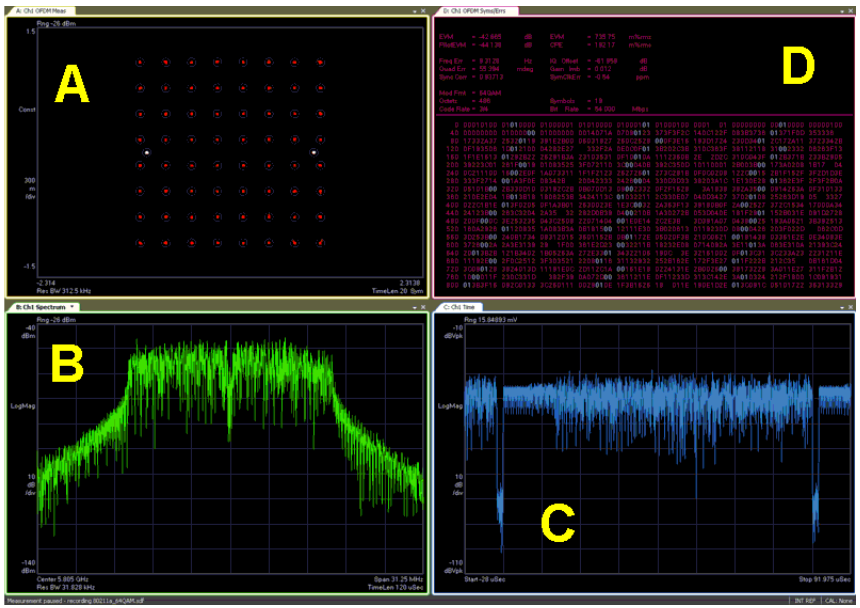
16-point constellation

Now suppose we put four points in each quadrant.
This gives us a system where, during each cycle, we can hit any one of sixteen points, which means that, with each cycle, we can transmit a 4-bit word. A dot pattern of this type is called a **constellation** because, when it's displayed as white dots on a black background, it resembles a star pattern.

This is splitting hairs, but many people insist that a modulated *phasor* is not a phasor but a *vector*. “Phasor” has a specific meaning: a phasor is a vector with an unchanging length, rotating at a constant speed in a counter-clockwise direction, representing a sine wave at a particular frequency. As soon as you modulate it, it's no longer a phasor. I am not in a position to settle that argument; I'll just call it to your attention so when you hear other techs arguing about it, you'll know it's time to quietly leave the room for a while.

THE DATA ANALYZER

There are all kinds of analyzers built for special purposes, but I want to briefly mention an all-purpose one that we'll look at in greater detail in EM6. The all-purpose data analyzer has a display that is broken into four smaller displays. They all work together, displaying four aspects of the signal.



Going counterclockwise, “A” shows a constellation. The possible points are in red and the actual signal is white.

“B” is a spectrum analyzer plot. We can see that the signal contains energy across a well defined range of frequencies, and that the energy drops off sharply on either side. It is possible that there is *no* energy outside the main band, and that the slopes are measuring artifacts. Once again, one must be intimately familiar with the equipment.

“C” is a regular time-vs-voltage oscilloscope display. In both the time and frequency displays, the signal resembles noise. It is more probable that we're dealing with a lot of different signals, e.g. cellphone channels, and their random combining results in a noise-like display.

In “D,” the data has actually been decoded into hex numbers. (Image from Wiki Commons)

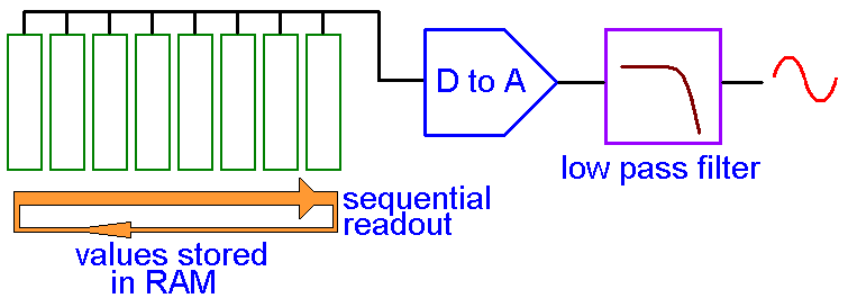
A data analyzer like this costs an arm, a leg, and two wisdom teeth. You'll probably never own one, but you may work with one your employer has bought.

DIGITAL WAVEFORM GENERATORS

We've been looking mostly at ways to analyze existing signals, but there's a lot to be said for generating signals yourself. For one thing, then you know exactly what the signal is supposed to be.

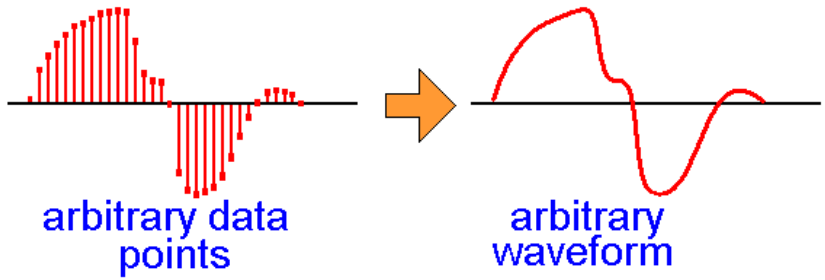
Some specific-purpose generators calculate waveforms on the fly. For example, D-flipflops and a few XORs can be wired together to create good pseudorandom noise.

But nowadays, most digital waveform generators use stored values in lookup tables. For example, one chunk of memory may be devoted to a sine wave. The program simply reads those values out of memory, looping through again and again. If 24-bit samples is used, waveform distortion can easily be less than 1 part in 10 million.



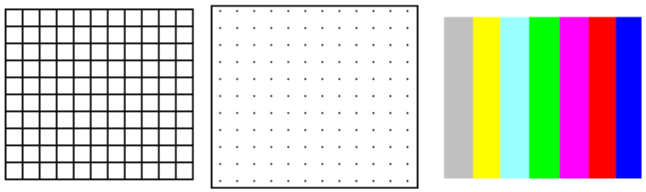
The diagram above shows how a digital signal generator might use a lookup table to generate a sine wave. Of course, the output of the D-to-A would be “bumpy,” so a low-pass filter is used to smooth things out and create a nice analog wave. A good digital generator built this way will give even the best analog generator a run for its money in terms of signal quality and stability.

The real advantage of lookup tables is that it's just as easy to define waveforms that do not correspond to the traditional waveshapes. If you like, you can choose the voltage for each sample point in the wave, and come up with something completely arbitrary. This kind of generator is known as an **arbitrary waveform generator**.



The better generators will give you some assistance. You can choose the shape of the overall wave, or define frequency and phase relationships, or define the waveform according to harmonic content, or specify a bandwidth, and the generator will figure things out for you. This is where a good knowledge of the science of harmonics and bandwidth comes in handy.

With today's complicated signals, a signal generator is often called upon to reproduce a complicated pattern specifically designed for testing a particular function. Actually, this is nothing new. Back in the days of CRT televisions, a tech would carry a test generator that would generate a dot pattern, a color pattern, and several other patterns. They even used to build those things with tubes – was *that* ever complicated!



video test patterns

You don't see video test generators much anymore because LCD panels don't go out of alignment. Video projectors sometimes require them if they have separate red, green and blue imaging elements that must be adjusted for proper convergence.

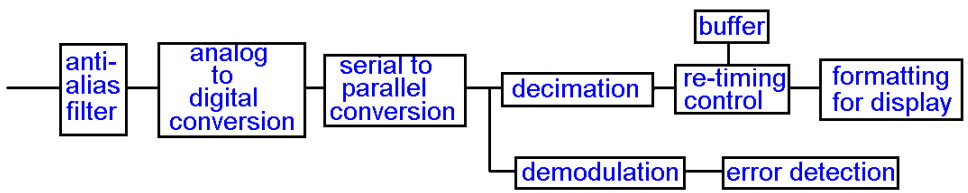
What you do see digital generators doing, besides generating the standard waveforms, is generating specific logic sequences supplied by a manufacturer and designed to help with installing and calibrating that manufacturer's equipment. When possible, the data is organized so it will show on a data analyzer's display in a pattern that makes a defect call attention to itself, like a missing button on a shirt.

For example, in EM6, you're going to discover that such things as wireless microphones and cellphones have extremely complicated signals, to the point where it's hard to tell whether a pattern on an oscilloscope or data analyzer represents a malfunction or normal operation. Arbitrary waveform generation can be used to mimic these signals, but with a known pattern that allows the tech to tell when something is wrong.

BLOCK & SCHEMATIC DIAGRAMS

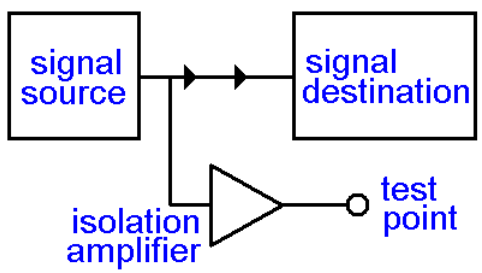
The “road map” of a unit or system is the diagram supplied by the manufacturer or installer. The two most common diagrams are **block** diagrams and **schematic** diagrams.

In block diagrams, the various functional parts are shown symbolically, as boxes. The symbols correspond to actual circuits, but when you first set out to find a malfunction, the complexity of part-by-part diagrams can make it hard to see the larger picture. The symbolic approach, which clearly divides a complex machine into functional chunks – “at this point, *this* happens to the data, then at that point, *that* happens” – permits the tech to quickly pin down the general area where the problem is occurring.



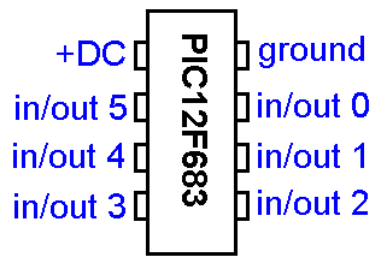
If you're lucky, the diagram will show proper voltages, waveforms etc at the interconnection points between blocks. If you're really lucky, the equipment will include test points. You can connect a scope or analyzer to any point in the circuit, of course, just by putting a probe on it. But, as I've already mentioned, simply attaching a probe can screw up system operation. You always have the loading and grounding problem and frequently have an access problem – for example, the connections to a high-density IC are often *under* the chip, where there may be no way you can get at them. However, the diagram may indicate another point where you can make a connection to that same part of the circuit.

When a block diagram indicates a test point, and especially if the unit has an actual test terminal permanently wired into the circuit, that means the manufacturer considers that an appropriate point to measure from. Often the test point will have a buffer amplifier which is not there to boost the signal, but to isolate your test unit from the circuit to prevent loading.



Once you've located the general area of the malfunction, it's time to turn to the schematic diagram to find out just which part is stinking up the system. Sometimes the “stinking” is literal – you may find a part that smells like burnt plastic, or the part may have been so hot for so long that the PC board under it is discolored. The part may even have a visible crack in it. But Murphy won't let you off that easily very often.

Schematic diagrams show actual parts. When we're talking about digital circuitry, many of those parts are themselves black boxes. For example, an MPU will be shown as an IC with pins and connections labeled. The circuitry inside the MPU may be a mystery to everyone but its manufacturer. But there is still a level of detail the block diagram doesn't show. All the connections to all the surrounding components will be drawn out, and a savvy tech who has downloaded the MPU's data sheet can poke around and figure out where the culprit is hiding.



BUILD-FOR-TEST AND SELF-TEST

With modern digital equipment, servicing has become so complex that manufactures have started helping the poor tech out by moving some of the test functions off the bench and into the equipment itself.

As digital (and analog) equipment works with ever higher frequencies and data rates, this tendency toward self-test is accelerating. One common example is a test routine in firmware that automatically exercises every aspect of every function of the machine and reports any problem. Car manufacturers have included self-diagnostics of this type for years.

SYSTEM-LEVEL SERVICING

Okay, let's put the enclosures back together, put in the screws, and deal with digital technology the way many techs deal with it: as black boxes. What do you do when you have a system with lots of interconnected units, and something isn't behaving?

What you don't do is just blindly start swapping parts of the system to see what will work. With that approach, it could take you a long time to find the malfunction. Or, you might never find it at all, if the problem is caused not by a piece of defective equipment but by incompatibilities between pieces of equipment.

I'm just going to do a quick summary here; we'll be talking more about servicing in EM5. But there are a few things that are specific to digital systems.

If you're servicing a system, rather than a particular unit, you will likely never get to the schematic diagram stage. You will be examining the data at the inputs and outputs of various units, and perhaps at the software, and when you find something defective you'll just swap it out. Some other tech at the defective unit's manufacturer will do the actual fixing, if fixing it is worthwhile.

The big problem here is that the person who draws up the block diagram of a system is the installer, and many installers don't bother – or the diagrams aren't updated when system changes are made. Sometimes a customer doesn't even know a diagram exists. In this case it may be worth your while to trace out the system and draw your own diagram. Especially in a complicated system that covers, say, a whole building, that investment can pay off. After you've finished, it's a good idea to make a complete system diagram, and take one copy to put in your own files.

ASK QUESTIONS!

When you arrive at the customer's site, the first thing you do is ask questions. When did the system go down? Have there been any recent system changes? Has any building remodeling been done? Have you had other contractors doing work?

There are three certainties in life. One is death. The second is taxes. The third is that if an electrician, carpenter or plumber is doing work in the building and an Ethernet cable is in their way, they'll chop it in two and not bother telling anyone. They will probably get a perverse thrill from it, as they're envious of your electronic smarts. They may even seal the ends up behind new drywall. By asking questions, you can often pin down a problem in five minutes that would take you hours to find otherwise.

ISOLATING THE PROBLEM

The next task is to isolate the problem, to find out where it's occurring. If the system is an all-in-one package like a laptop, at least you know the problem is inside the laptop. When you have a lot of different pieces of equipment connected together, the problem may be in a particular unit, or it may be in the interconnections. Most techs start with the interconnects, just because it's easier to fiddle with them than with the insides of a piece of gear.

If the system has multiple units of the same type, try swapping them. If the problem moves when the units move, you can fairly confidently pinpoint the defective unit.

What if swapping units doesn't solve the problem on the first try? Don't do a second try – don't try another swap of the same type. With the first swap, you've established that the unit you're swapping isn't the problem. Try swapping something else, but first put the system back to its original condition.

If, once you've put the system back the way it started, something that worked before doesn't work now, you've very likely got an interconnection problem, and you've probably burned out the second unit. Time to get out the voltmeter and make sure DC at the connector is on the pins it's supposed to be.

POE PROBLEMS

A special warning about **POE** or **power over Ethernet** systems. In chapter six, I mentioned that most of the time, an Ethernet link uses two of the four pairs of a CAT5/6 cable, and the other two pairs are often used to supply power to a unit. The problem here is that there is no universally accepted standard regarding those power wires. For example, if you try to intermix Netgear and Sysco products, you'll likely burn something out, since their POE standards are not the same. Be very careful when using POE. You need to either:

- study manufacturers' literature and know exactly how each one handles POE, which may require you to rewire cables, or
- get all the equipment from one manufacturer.

If you're having trouble getting a POE-based system to work, after about the second failure you should stop swapping units in and out. You may very well be damaging something with each swap. The problem either lies with POE incompatibility, or in some completely unrelated area.

SOFTWARE DIAGNOSTICS

It's a waste of time, and often impossible, to devise extensive equipment tests on your own. Equipment manufacturers usually offer complete libraries of diagnostic software you can download from the internet for free, and their tests will be better than yours because they have probably seen your specific problem before and built a test for it into the software.

Diagnostics, though, may not solve problems caused by incompatibility between different units, especially from different manufacturers. The number of possible system variations is nearly infinite, and there's no way a manufacturer can test for every combination. Likewise, there may be no way a manufacturer's tech support people can help you if the problem occurs between a piece of their equipment and another piece made by somebody else.

Crowdsourcing to the rescue: you're likely not the first person to have a particular problem. Go online and search. And when you find a solution, don't stop, keep searching. The first solution may not be best, or may come from someone who doesn't really know what they're doing, and their solution only worked by accident. By the time you're done with a thorough search, you will have several likely solutions, several warnings of things not to do, a deeper understanding of the equipment you're dealing with, and some extra things to put in your bag of tricks for the future. This is one of the ways average techs become wizard techs.

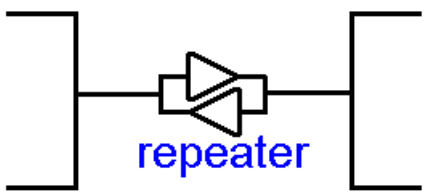
REPEATERS

If two units are connected by a long cable, and data isn't getting from one to the other or the error rate is suspiciously high, try moving the units closer and connecting with a shorter cable. Cable attenuation is a very common cause of system malfunction. (With any luck, that diagnostic software you're running will tell you without your having to move things around.)

This is one of those areas where good cables are definitely better than cheap cables. There is an actual, measurable difference between Belden cable and no-name import stuff. I once had to add an equalization circuit to a piece of equipment I was designing, specifically to compensate for this difference.

When you're putting a system together, or troubleshooting somebody else's system, keep in mind the maximum specified length for the type of link you're dealing with. Up to about ¼ of that length, cheap import cables should work. Between ¼ and $^{2}/_{3}$, it's better to use quality American-built cables. Near maximum length, only the best will do.

If you're dealing with an already-installed system with cable-loss related problems, the easiest thing to do is chop the cable in the middle and install a **repeater**, which is an amplifier that boosts the signal en route. Many repeaters also offer equalization to partially offset the waveform smearing that comes with high frequency attenuation. It's a bandage approach, but it works. Repeaters are often powered by the cable – if you're dealing with an Ethernet system, use caution. Otherwise, they will be powered by "wall wart" plug-in transformers. This adds a new potential for failure, because somebody who doesn't know what that wall wart is for may unplug it. Try to find a place to insert the repeater where the AC power will always be on and access by random personnel will be limited



Hey, guess what? We're done! Well actually no, you're just done with the first part of your training, the part that gets you to where you can pass the EM4 exam. I'm past 60 and I still consider myself "in training." Any tech who doesn't is in the process of making him/herself obsolete. But at this point a modest celebration is in order. Like, go back and read the first few pages of the EM1 study guide. See how far you've come?

End of chapter 10

CHAPTER 11 – EM4 REVIEW

CHAPTER		PAGE
1	Digital Concepts	2
2	Binary Arithmetic	3
3	Boolean Logic	6
4	Sequential Logic	8
5	Microcontrollers and Programming	11
6	Computers	15
7	Mixed Digital / Analog	18
8	Machine Control	21
9	Displays and Printers	25
10	Servicing Digital Equipment	28
---	Charts & Graphs	32

CHAPTER 1 – DIGITAL CONCEPTS

DIGITAL – refers to a system where all signals take the form of discrete, specific levels, usually measured in terms of voltge. There can be any number of levels, but most commonly the number of levels is two: a high level and a low level.

DIGITAL ADVANTAGES –

- chunky, not degraded by small distortions, robust
- fits computers
- can process signals in ways analog cannot

DATA – what a digital signal is usually called

HIGH – the higher of two voltage levels, also called "**1**."

LOW – the lower of two voltage levels, also called "**0**."

VOLTAGES – binary 1 is represented by a voltage, traditionally +5V; 0 is represented by 0V.

BINARY – a system of mathematics where there are only two numbers, 1 and 0. Usually taken to mean any system that computes or calculates using two levels.

LOGIC – the process of digital computing, and the circuitry that performs the computations.

PULL UP – what a circuit does when it connects an output to the +DC supply, "pulling" the output voltage up to the supply voltage.

PULL DOWN – what a circuit does when it connects an output to ground, "pulling" the output voltage down to ground potential.

LOGIC FAMILIES – the general design structure of digital circuits. These include:

RTL – resistor-transistor logic, totally obsolete
DTL – diode-transistor logic, totally obsolete
ECL – emitter coupled logic, a high-current, high speed family now obsolete
TTL – transistor-transistor logic, obsolete but found in older equipment
CMOS – complementary metal-oxide semiconductor, the present-day technology

FANOUT – the number of paralleled logic gate inputs that a logic gate output of the same family can drive.

COMPLEMENTARY – a circuit that uses both NPN and PNP bipolar transistors, or N-channel and P-channel field-effect transistors, to pull an output up and down. This is in contrast to TTL, which uses a transistor to pull an output one way and a resistor to pull the other. Complementary circuitry exerts a firm "pull" in both directions.

TTL POWER SUPPLY VOLTAGE – +5VDC, always, no exceptions.

Review page 2

STANDARD CMOS SUPPLY VOLTAGE – standard CMOS will function with supply voltages from +3V to +18V. Other kinds of CMOS, especially those with a "T" in the number, work best at +5V or lower.

ACTIVE HIGH – the most common type of input or output, where the higher of the two voltages (e.g. +5V) represents "1."

ACTIVE LOW – in some circuits, the lower of the two voltages (e.g. ground) represents "1." Control and enable inputs of many digital ICs are active low.

UNDEFINED VOLTAGE – a voltage that is not close to either the +DC supply or ground. The circuit may be unable to decide whether it represents a 1 or a 0. For TTL, any voltage between about 0.8V and 2V is undefined. For standard CMOS, any voltage that is not within 1.5V of one supply rail or the other is undefined. Low-voltage CMOS has tighter requirements.

FLOATING INPUT – an input left unconnected. Bad practice with TTL. With CMOS, such bad practice that the circuit will not work and may destroy itself.

SHOOT-THROUGH – when both transistors of a complementary pair are "on," creating a direct current path from the power supply rail to ground. CMOS circuitry often uses this effect to speed up transitions, but if the circuit stays in shoot-through too long, it may be destroyed. This condition is caused by an input in the undefined voltage region. Thus an input must never be allowed to drift into the undefined region; all inputs must have a current path that pulls them either up or down. A CMOS input should never be left unconnected.

ESD – electrostatic discharge. CMOS circuitry is easily destroyed by ESD, so techs working on CMOS must take steps to avoid it (see chapter 10). Precautions include:

- build-out resistors in series with inputs and outputs
- snubber diodes between inputs and ground, and inputs and +DC
- grounding tools, workbench and technician when working on equipment.

PART NUMBERS –

- 74XX TTL
- 74LSXX low power schottky ttl
- 4xxx standard cmos
- 74CXX ttl design converted to cmos
- 74xCTxx cmos with ttl-compatible voltages

  Letters after the number denote case and other details

  Some ICs are given "house numbers" specific to the equipment they're used in

CHAPTER 2 – BINARY ARITHMETIC

DIGIT – a single number, like 0, 1, 2, 3, 4, 5, 6, 7, 8 or 9.

Review page 3

POSITIONAL NUMBER SYSTEM – a number system where the position of a digit signifies its value. The “1” in 1,000 is ten times more significant than the “1” in 100.

BASE, RADIX – the set of numbers used in a number system.

DECIMAL – a number system based on ten digits, 0 thru 9. This is called a base 10 system.

MOST SIGNIFICANT DIGIT – in a multi-digit number, the digit farthest to the left, which has the greatest “weight.” In the number 1,234 the most significant digit is “1,” representing a value of 1,000.

LEAST SIGNIFICANT DIGIT – in a multi-digit number, the digit farthest to the right, which has the least “weight.” In the number 1,234, the least significant digit is 4.

BINARY – a number system based on two digits, 0 and 1. This is called a base 2 system.

BINARY PLACES – in binary, positions to the left of the point represent powers of two: ones, twos, fours, eights, sixteens and so on. Positions to the right represent powers of ½: half, fourth, eighth, sixteenth and so on.

MOST SIGNIFICANT BIT – in a multi-bit number, the bit farthest to the left.

LEAST SIGNIFICANT BIT – in a multi-bit number, the bit farthest to the right.

PLACE HOLDER – a zero inserted to “hold” a place, so digits to the left will keep their proper significance. In the number 1,000 the three zeroes are what tell us the “1” means one thousand.

WHOLE NUMBER – a digit representing a whole quantity, as in 1, 2, 3 and so on. Also called an **integer**.

FRACTIONAL NUMBER – a digit representing a quantity less than one, as in 0.3.

POINT – in a multi-digit number, the dividing point between whole numbers to the left and fractional numbers to the right. Place holders can be inserted if needed to hold the positions of numbers to the right of the point. If we're working with decimal numbers, the point is called the **decimal point**. In binary, it's just called the point.

BINARY ARITHMETIC – addition, subtraction, multiplication and division tables similar to the ones used with decimal arithmetic, but much simpler because there are only two possible values, 0 and 1. A complete table is shown in the charts and graphs pages at the end of this review.

BIT – a binary digit, 1 or 0.

BINARY WORD – several related bits that represent a number or other data.

Review page 4

BYTE – an 8-bit word. Typically abbreviated capital-B to distinguish it from a bit, which is abbreviated small-b.

NIBBLE or **NYBBLE** – a 4-bit word.

OCTAL – a number system with a radix of 8. Possible digits are 0, 1, 2, 3, 4, 5, 6 and 7. An octal digit can be represented by three binary bits, e.g. $101_2 = 5_8$.

HEXADECIMAL – a number system with a radix of 16. Possible digits are 1,2,3,4,5,6,7,8,9,A,B,C,D,E and F. A hexadecimal digit can be represented by four binary bits, e.g. $1101_2 = D_{16}$. Hexadecimal is often abbreviated **Hex** and represented by a dollar sign in front of a number, e.g. **$F**.

BINARY CODED DECIMAL – BCD, a number system that represents decimal values using bits. Possible values are 0_{10} thru 9_{10}, 0000_2 thru 1001_2. On the next count, 1001 resets to 0000 and a 1 is carried to the next significant digit.

FLOATING POINT – a kind of binary math where the position of the point is allowed to move back and forth to handle very large or very small numbers. This permits tiny errors which are unacceptable in some situations.

ROUNDING – different from the rounding rules laid out in EM1, this is just a convenient way of talking about big digital numbers.

- A 10-bit digital number has 1,024 possible values, but we refer to it as “1k.”
- 16 bits have 65,535 possible values, but we call it “64k.”
- 20 bits have 1,048,576 possible values, just over 1 million. We call it “1meg.”
- 1,024 megs are “1gig,” as in gigabyte.
- 1.024 gigs are “1ter,” as in terabyte.

ALPHANUMERIC – a coding system that assigns binary words to all the decimal number digits, all the English letters, and common punctuation symbols

EBCDIC – a proprietary IBM alphanumeric system that encodes numbers, letters and computer instructions into 8-bit words. Still used because there is so much existing data coded in EBCDIC.

ASCII – Extended Binary Coded Decimal Interchange Code, the standard open, non-proprietary system that encodes numbers, letters and instructions into 7-bit words. It is the coding standard of the English-speaking world. ASCII and EBCDIC are not compatible, though with today's technology, translating between them is simple.

UTF-8 – an extended encoding system using 8-bit words and including all the standard ASCII characters, plus characters used in other languages.

UNIVERSAL CHARACTER SET – the eventual goal of UTF-8 is to include all the characters used in every language.

Review page 5

UTF-16 and **UTF-32** – more complex standards not compatible with ASCII.

CHAPTER 3 – BOOLEAN LOGIC

BOOLEAN LOGIC – a logical system based on two possible states: something is true, or something is not true. Boolean logic can be built into binary circuitry by defining "true" as 1 and "false" as 0. Also called **combinational logic**.

GATE – an electronic circuit that embodies and performs a Boolean function.

A, B, Y – A and B are logic inputs, factors which may be true or false. Y is the output, the result of the Boolean operation.

BASIC BOOLEAN FUNCTIONS

- NOT – if A is true, y is false.
- OR – if A or B or both are true, Y is true. If A and B are false, Y is false.
- AND – if A and B are both true, Y is true. If A or B or both are false,Y is false.

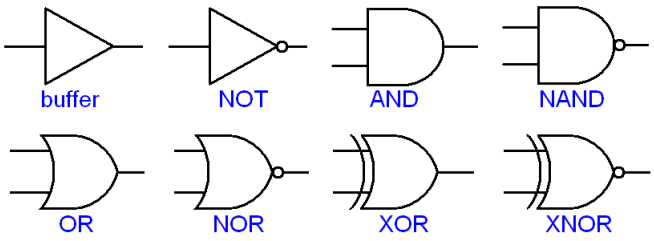
All possible Boolean questions can be answered by building logical sequences using these three functions. "Building" may mean actually constructing circuits using the appropriate gates, which will return a "1" or "0" answer.

There is disagreement over whether a NOT should be called a gate. It does not perform a decision function, but the NOT function it performs is one of the basic Boolean functions.

ADDITIONAL GATES – there are standard gates that expand on the three most basic functions, permitting logical sequences to be built more conveniently.

- NOR – not OR, the same as an OR, but with an inverted output.
- NAND – not AND, the same as an AND, but with an inverted output.
- XOR – exclusive OR. If A and B are false, the result is false. If A or B are true, the result is true. If A and B are both true, the result is false.
- XNOR – exclusive NOR, the same as an XOR, but with an inverted output.

BOOLEAN GATE SYMBOLS – note that a small circle always indicates an inversion.



TRUTH TABLE – a chart that shows every possible combination of input states and the resulting output state for each.

Review page 6

GATE PACKAGES – an IC may contain several gates of one type, identified as follows:

- dual 2 gates
- tri- 3 gates
- quad 4 gates
- hex 6 gates
- octal 8 gates

BUFFER – a circuit that transfers input state to output state with no Boolean decision-making. a non-inverting buffer transfers input to output unchanged. An inverting buffer inverts the input on its way to the output, e.g. 1 becomes o. An inverting buffer is a NOT.

SCHMITT TRIGGER – a buffer that uses positive feedback to create a fast, decisive 0-to-1 or 1-to-0 transition. The input has hysteresis; the input voltage must rise to about 2/3 of the supply voltage to count as a "1" and fall to about 1/3 to count as a "0." This means the input has no undefined voltage region.

DATA BUS – a data path, one or more bits wide, for moving data around within a system.

BUS FIGHT – when two or more outputs try to send data over the bus at the same time. This is also called a **collision**.

TRI-STATE – an output with three states: 0, 1 and disconnected, indicated by "Z" for high-impedance. Its purpose is to avoid bus fights. At any given moment, only one of the outputs connected to the data bus is sending 1s and 0s. All the others are in the Z state, effectively disconnected.

LEVEL SHIFTER – a circuit that translates a low-voltage "1" to a higher voltage "1" – e.g. +5V to +12V – or vice versa

ENABLE – an input that activates or deactivates a function. One common use is to switch a gate between normal output and Z.

OPEN COLLECTOR / DRAIN – a gate whose output can only pull down.

WIRED AND, WIRED OR – a Boolean function created by wiring together several open collector / drain outputs

GATE DELAY – also called propagation delay, the time it takes for a state change at the input of a gate to reach the output.

WAVE SHAPING – manipulating waveforms so as to change their relative timing. Used to insure that events that occur close together happen in the right order.

BOOLEAN MATH AND NOTATION – standard symols and rules that permit Boolean logic to be performed mathematically.

$$Y = \overline{A}$$

Review page 7

- NOT – a line over a letter signifies inversion.
- AND – a dot. Mathematically, AND is a multiplication. The and function cal also be written Y=AxB or Y=AB. $Y = A \bullet B$
- NAND – the same, with a line. $Y = \overline{A \bullet B}$
- OR – a plus sign. Mathematically, OR is an addition. $Y = A + B$
- NOR – the same, with a line. $Y = \overline{A + B}$
- XOR – plus sign in a circle. $Y = A \oplus B$
- XNOR – the same, with a line. $Y = \overline{A \oplus B}$

COMPUTATION RULES – listed at the end of this chapter, print them out. Some are the same as the rules for regular math; some only apply to Boolean math.

SIMPLIFICATION – using math rules to reduce the number of terms in a Boolean equation. This translates to fewer actual gates in a circuit.

KARNAUGH MAP – a graphical way of finding the simplest math expression for a certain logical sequence.

CHAPTER 4 – SEQUENTIAL LOGIC

SEQUENTIAL LOGIC – logic operations with specific “before” and “after” states; logic tied to events in time.

STATES AND TRANSITIONS-

- FLIP – to change from 0 to 1.
- FLOP – to change from 1 to 0.
- FLIPPED – in the “1” state.
- FLOPPED – in the “0” state.
- SET – to flip, or be in the flipped state.
- RESET – to flop, or be in the flopped state
- CLEAR – same as reset.

ASYNCHRONOUS – an event that is not tied to any timing signal; also circuitry that operates in this manner.

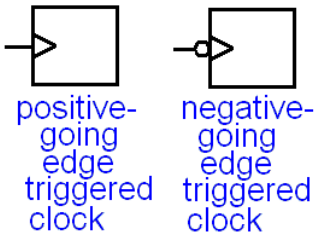
SYNCHRONOUS – an event that is tied to a timing signal, and does not happen until the timing signal tells it to; also circuitry that operates in this manner.

Review page 8

CLOCK – the timing signal that tells synchronous circuitry to operate. The word can mean a single clock event, or a repeating waveform that runs continuously.

POSITIVE-GOING CLOCK – the rising edge of a clock waveform, i.e. the transition from 0 to 1. In a schematic symbol, indicated by a triangle.

NEGATIVE-GOING CLOCK – the falling edge of a clock waveform, i.e. the transition from 1 to 0. In a schematic symbol, indicated by a triangle and the circle that signifies inversion.



PARALLEL – a system where all the bits of a word move together, on separate wires.

SERIAL – a system where the bits of a word move single-file, one after another, on a single wire.

LATCH – a circuit that captures and input state, transfers it to its output, and holds it in that state even after the input is changed.

Q = the "main" output of a latch or flipflop, similar to a Boolean Y. Not to be confused with the analog Q, quality of a tuned circuit

notQ = a second output which is exactly like the Q output, except that its polarity is reversed. That is, whenever Q is 1, notQ is 0. In text, often identified as nQ or -Q. In diagrams, identified as Q with a line over it.

LATCH – a circuit that "grabs onto" state 0 or state 1 and stays there until it receives an input that tells it to change. In computing circuitry, "latch" usually refers to a circuit that captures the state at its input at a certain moment and remains in that state, even after the input has changed.

SR LATCH – set / reset latch, a latch where "1" on the S (set) input sets Q, and "1" on the R (reset) input resets Q. Also known as a **SC** latch for set / clear. It is not permissible for S and R to both be 1 at the same time.

GATED LATCH – a latch with a single "D" (data) input. When the gate input is high, Q follows D. When the gate input goes low, Q is frozen in its current state.

CLOCKED LATCH – like a gated latch except that data is transferred from D to Q on the rising edge of the clock input. The clock must go low and then high again for another transfer to occur. The clocked latch is like a camera taking snapshots.

FLIPFLOP – a circuit that can be set up to alternate. With successive input pulses, Q goes high, low, high, low etc.

T FLIPFLOP – a flipflop that toggles, or alternates as described above.

Review page 9

D FLIPFLOP – delay flipflop, a clocked latch by another name. Data is delayed from input to output by one clock cycle.

SR FLIPFLOP – an SR latch with a clock input. The S and R inputs set and reset the output as with the latch, but wait for "permission" from the clock to do so. If S and R are both low, nothing happens when the clock occurs. S and R are not permitted to be high at the same time; the circuit designer must see that this does not happen.

JK FLIPFLOP - like an SR flipflop in most ways; when J is high, Q flips on the next clock, when K is high, Q resets on the next clock. However, J and K can both be high at once, in which case the JK flipflop toggles like a T-flipflop. Most JK flipflops also have set and clear inputs, which affect the outputs immediately.

SHIFT REGISTER – an array of D flipflops that transfers an input bit pattern down the line, one stage at a time. The input data appears at the output, delayed by as many clock cycles as there are shift registers.

BINARY COUNTER – also known as a ripple counter, a circuit that counts in binary: 0, 1, 10, 11, 100 and so on. A string of T flipflops with each notQ connected to the next flipflop's clock input makes a binary counter.

UP-DOWN COUNTER – a binary counter with a control input that can make the counter count either up or down.

PRESETTABLE COUNTER – a counter that can be pre-loaded with a binary number, and then count up or down from there depending on the state of its up / down control input

RING COUNTER – a shift register where the output is connected back to the input so the bit pattern circulates.

SERIAL IN, PARALLEL OUT SHIFT REGISTER – a shift register where the output of each stage is connected to a terminal, so you can get at it. The outputs of all the stages form a binary word that shifts down the line one position on each clock.

PARALLEL OUT, SERIAL IN SHIFT REGISTER – a shift register where a parallel binary word can be pre-loaded into the register. Used to convert parallel data to serial data.

MASTER-SLAVE – a circuit design where data is transferred in two steps, insuring that a transition only occurs on a clock and only one transition can happen at a time.

MODULUS COUNTER – a counter that is designed to count up to a certain number, e.g. 5, 7, 13, 144, and then reset to zero and start the count over.

BCD COUNTER – a counter with a modulus of 10. It counts up to 1001_2, then on the next count resets and begins counting from 0 again.

Review page 10

CHAPTER 5 – MICROCONTROLLERS AND PROGRAMMING

MICROCOMPUTER or **MICROCONTROLLER** – a small computer in a single IC package, with all the input, output and computation functions necessary to do a job. Abbreviated **MCU**.

EMBEDDED – an MCU that's part of another machine.

HARDWARE – physical computing machinery like an MCU or laptop computer.

SOFTWARE – a set of instructions that a computer runs.

FIRMWARE- software that is permanently installed and intended to last as long as the computer.

PROGRAM – also called “code,” the instructions that tell the MPU what to do.

CPU – central processing unit, containing the ALU and all support circuitry immediately connected to it.

ALU – arithmetic / logic unit, where computation is done

INSTRUCTION DECODER – a circuit that converts binary instruction codes from the program into actions which the MCU executes.

ACCUMULATOR – a small “scratch pad” memory used by the ALU to hold data that it's currently working with.

PROGRAM COUNTER – a counter that keeps track of which instruction is to be executed next.

STACK POINTER – holds the address of the next instruction. Used when the program jumps to a different address; the pointer tells the program where to return to.

CONDITION CODE REGISTER – monitors such things as whether the result of a computation is zero, or negative, or whether there is a carry or borrow.

TIMER – used to time various functions.

WATCHDOG – a timer that alerts the system if program execution has stopped for more than a preset length of time

MEMORY – binary information storage, not part of the CPU.

RAM – random access memory, where data can be stored (write) and retrieved (read) as often as desired.

Review page 11

ROM – read-only memory, which holds the program and any unchanging data. Information is placed in ROM once, before the MCU is put into use, and cannot be changed. Some MCUs have the option of destroying the pathway into the ROM after the program is loaded to be sure it cannot be changed or examined. Most MCU memory is ROM.

BURNING – loading a program into ROM.

VOLATILE – memory that loses its data when power is turned off. RAM is volatile.

NONVOLATILE – memory that keeps its data when power is turned off. ROM is nonvolatile. RAM can be made nonvolatile by providing a backup battery or large capacitor.

MASK PROGRAMMING – a process where the MCU's program is loaded into ROM at the factory photographically. Used in large-quantity applications.

PROM – programmable ROM that allows the user to load the program.

EPROM – ROM that can be erased by UV light and reloaded, usually used in program development.

EEPROM – EPROM that can be erased electrically.

EAPROM – electrically alterable ROM where changes can be made to the code stored in individual addresses, rather than erasing and reloading the whole ROM.

ASIC – Application Specific IC. An IC manufactured to serve one specific purpose. Most often an MCU does only one thing all its life. When it is new, a program is stored in it to tell it how to do that one thing, and from then on it's essentially an ASIC.

MEMORY MAP – a chart showing how various memory addresses are assigned, e.g. which addresses are RAM and which are ROM.

BUS – wires used to move data and instructions around.

PORT – an interface that allows the MCU to communicate with the outside world. In many cases a port is memory mapped, meaning that the MCU treats it like a location in RAM.

DATA DIRECTION REGISTER – determines whether each bit of a port is incoming or outgoing at a particular moment.

INSTRUCTION SET – a list of instructions an MCU can perform.

Review page 12

RISC – reduced instruction set, an instruction set that's deliberately limited for the sake of simplicity and low cost. Some common instructions:

- arithmetic functions
- Boolean functions
- gate, latch and flipflop functions
- bit test functions (tests whether a bit is 1 or 0)
- bit and word shift and manipulate functions
- reading data from RAM and writing data to RAM

SYSTEM CLOCK – a continuous squarewave signal that controls the timing of all functions.

MACHINE CYCLE – the time it takes for the MCU to do one thing, which might be likened to one rotation of a motor. A machine cycle may be one clock cycle or several.

INSTRUCTION CYCLE – the time it takes to execute an instruction, measured in machine cycles. To maximize speed, instructions are written so as to take as few machine cycles as possible.

PIPELINING – speeding up execution by fetching the next instruction while the current instruction is being executed.

HARVARD ARCHITECTURE – a design with two buses, one for data, one for instructions.

VON NEUMANN ARCHITECTURE – a design where data and instructions take turns using a single bus.

FLOW CHART – a visual description of a program, using different shapes to identify different kinds of events.

UNCONDITIONALOPERATION – a command that is simply executed in sequence. In a flow chart, represented by a rectangle.

CONDITIONAL OPERATION – an operation that asks a question, and the answer determines what happens next. Represented by a diamond, with the points corresponding to different possible answers. Usually, there is a “normal” outcome and an “abnormal” one. When an abnormal outcome is encountered, the program will **branch** off in a different direction to handle the exception.

JUMP – where the program counter stops following instructions in the normal order and jumps to another point in the program.

SUBROUTINE – a small sub-program that may be called upon by the main program. When the subroutine is finished, it may tell the program counter to restart the main program at a certain point. Or, it may tell the MCU to return to where it was before the jump. That information is contained in the stack pointer.

LOOP – a section of program that repeats, e.g. decrementing to zero, one step at a time.

Review page 13

INTERRUPT – causes the MCU to stop its normal program execution and go to a special section of the program, usually a subroutine, to deal with the interrupt. This is called **servicing** the interrupt.

HARDWARE INTERRUPT – an interrupt generated when external circuitry pulls the interrupt pin high (or low).

SOFTWARE INTERRUPT – an interrupt generated inside the MCU by, for example, the watchdog timer.

MASK – a flag set inside the MCU to disable the interrupt function when the CPU is doing something that should not be disturbed.

NONMASKABLE INTERRUPT – an interrupt that overrides the mask flag. Used for interrupt conditions that cannot wait.

DEBOUNCE – a common subroutine. When mechanical switch contacts close, they typically "bounce," meaning they close and open many times in the first few milliseconds before closing for good. A debounce subroutine is required to keep the MCU from interpreting every bounce as a separate closure.

LOOKUP TABLE – a set of memory addresses containing related data. The program may tell the MCU to find answers to a computation, for example, by going to the lookup table instead of actually doing the computation. This can often be faster.

MACHINE LANGUAGE – the instructions in an MCU's program written as binary words, 1s and 0s, which the MCU knows how to interpret. An MCU requires a machine-language command for every single thing it does, thus a machine language program is very detailed.

ASSEMBLY LANGUAGE – instructions in the form of English words or word fragments that are easier for a human to understand than machine language. The programmer writes using the assembly language, and the instructions are then converted to machine code.

HIGH LEVEL LANGUAGE – a language that describes larger-scale functional processes rather than individual instructions. A single high-level instruction might describe an action that takes dozens of machine instructions to accomplish. Since high-level languages focus on results rather than the process, they are not tied to a specific MCU family.

COMPILER – a program that translates high level code to machine language. A particular compiler will only work with one family of MCU.

INTERPRETER – a real-time compiler. Programs run more slowly with interpreters because the compiling must be done while the program is running.

Review page 14

UART – universal asynchronous receiver / transmitter, a part of some MCUs that reformats data as it enters or leaves the MCU so it can communicate with other equipment. This is different from a port, which inputs or outputs data in the parallel form already present inside the MCU, but does no translation or other processing.

HANDSHAKING – a protocol used by the UART to insure that data is transferred successfully.

ANALOG-TO-DIGITAL, DIGITAL-TO-ANALOG, COMPARATOR, PWM – ways of translating signals between the analog and digital worlds, covered more fully in chapter 7.

CHAPTER 6 – COMPUTERS

COMPUTER – an MCU on steroids. The essential difference is that the program that runs an MCU is stored once, in ROM, to do a single thing, while a computer stores programs in RAM and in other types of read/write memory, so as to be completely flexible in how they it is used.

MPU – Microprocessor Unit, the main computing IC in a computer. An MCU differs from a typical MPU in that the majority of the computer's memory is external to the MPU, and in most cases it runs much faster and has much broader capabilities than an MCU.

CORE – informally refers to a CPU. Computer makers furnish computers with multiple cores to increase speed.

POWER SUPPLY – a unit that converts 117VAC line voltage to the multiple DC voltages used by a computer, typically +12V, -12V, +5V and often +3V.

MOTHERBOARD – the main PC board in a personal computer. Every part of the system is mounted on the motherboard or connected to it.

DAUGHTER BOARD – a smaller PC board plugged into the motherboard via an EXPANSION SLOT. Daughter boards are commonly used to expand a computer's audio and video capabilities.

COPROCESSING – using separate ICs to handle some of the tasks the MPU would otherwise perform. For example, a math coprocessor IC may handle large number-crunching tasks.

MAIN MEMORY – the large block of memory that stores program and data. Most of this is RAM; there is very little ROM in a computer.

DYNAMIC RAM – memory which must be REFRESHED every few milliseconds to prevent its data from being erased. STATIC RAM, on the other hand, retains data as long as DC power is supplied to the ICs.

Review page 15

CACHE – a block of RAM inside the MPU, much larger than the memory in an MCU but a tiny fraction of the size of a computer's main memory. Used by the CPU for temporary storage of the data it's working with at a given moment. This partly avoids the main speed bottleneck, which is the data bus between main memory and the MPU.

OPERATING SYSTEM – the program that runs the computer, e.g. Windows 7 or Mac OS10.

BOOTING UP – short for bootstrapping. When the computer is first turned on, it uses the small program stored in ROM to read the much larger operating system program from the hard disc drive and store it in RAM so the computer can function.

HARD DISC DRIVE – essentially magnetic RAM; stores data on rotating discs coated with magnetic recording material. A hard drive can store much more data than the computer's RAM; the main drawback is comparatively slow access.

OPTICAL DRIVE – uses a CD or DVD that stores data as microscopic pits in an otherwise reflective metal layer.

SSDD – Solid State Disc Drive. A block of RAM so large that it can be used for the same purposes as a disc drive.

ERROR DETECTION AND CORRECTION – methods by which corrupted data can be repaired. Data stored on magnetic and optical discs always contain errors that require correction.

PARITY BITS – a basic error correction tool, bits added to data words to indicate whether they are supposed to sum to an even or odd number. Most data communication standards also use parity bits.

PORTS – connection points where data can be fed into and out of the computer. The connections for keyboard, mouse, monitor, printer etc are also considered ports, even when they carry analog signals.

SIMPLEX – a one-way communication channel. Data is sent from the computer to a machine or other receiving unit; there is no provision for reverse communication.

HALF DUPLEX – a two-way communication channel where the units on either end take turns sending data back and forth. A pair of walkie-talkies is a half duplex channel.

FULL DUPLEX – a two-way communication channel where the units on either end can both send and receive at the same time. A land-line telephone system is a full duplex channel.

DB9, DB25 – D-shaped connectors with 9 or 25 pins

8P8C, RJ45 – modular 8-pin connector similar to telephone connectors

Review page 16

NODE – a unit connected into a multi-station data channel. There are several standard cabling configurations:

- **STAR** – each node is connected by its own cable to a central distribution point called a **HUB**. The system resembles a starfish.
- **MULTIDROP** – a single cable is run to all nodes. Resistive termination at the end of the cable may be required.
- **DAISY CHAIN** – a cable is run from node A to node B, another cable is run from node B to node C and so on. The signal passes through circuitry in each unit that "repeats" the signal on the way to the next one.

UNSHIELDED, UNBALANCED – a single wire with a signal on it, measured against a ground or common wire. The cable's characteristic impedance is usually not specified. Often used with RS232.

UTP – Unshielded Twisted Pair, used by most of the newer communication standards. The signal appears as a voltage difference across a pair of wires. Usually about 100 ohm impedance. Used with most of the newer, faster digital interfaces..

COAX – coaxial cable, a wire enclosed in a braid or foil shield. The wire in the center carries the signal, measured against the shield. 50 ohms unless it's carrying video, in which case it is 70 ohm cable.

TWINAX – a twisted, balanced pair of wires inside a shield. About 110 ohms.

LAN – Local Area Network, a data network within, say, an office that connects computers, printers etc. Often wired using UTP.

WAN – Wide Area Network, a data network that connects, say, different buildings on a college campus. Due to high-frequency attenuation in cables, which increases with distance, wiring methods that work with LAN do not work with WAN. Must be wired with coax, twinax or optical fibers.

RS232 – a one-to-one data link that originated with teletypes. +3-25V = 0, -3-25V = 1. Full duplex, though the system can be run simplex if two-way communication is not required. Unbalanced signal measured against ground.

RS485 – a multidrop data link where the host queries each node in turn. Unbalanced multidrop cable, half duplex.

MIDI – Music Industry Digital Interface, a daisy chain system using a semi-balanced current loop. Optical coupling at each node eliminates ground potential problems. Simplex, but if the last node is chained back to the first, any node can communicate with any other. Primarily used with musical instruments and stage lighting, some machine control uses.

USB – Universal Serial Bus, intended for local area nets. Balanced UTP, star configuration. Four wires including the data pair and 5V and ground wires.

Review page 17

ETHERNET – balanced, multidrop system popular for LAN and WAN systems. Serial data. 10, 100, 1000BASE-T identify data rates of a bit less than 10Mb/S, 100Mb/S and 1Gb/S respectively. Voltages depend on data rate. Resistive termination at the end of the cable may be required.

FIREWIRE / IEEE 1394 – balanced, daisy chain system. Distance between devices is limited to 4.5 meters, but because the signal is "repeated" by every node in the chain, up to 63 nodes can be connected in a system with a maximum end-to-end length of 72 meters. Serial data, 2 pairs. Data rates of up to 3.2Gb/s currently, 6.4Gb/s is in development. IEEE 1394 is proprietary; manufacturers must pay to use it.

THUNDERBOLT – balanced, daisy chain system, serial data over four channels called "lanes." Maximum intended length was 3 meters and 7 nodes, but circuitry has reached such a degree of miniaturization that it's possible to build amplifiers right into the cable connectors. This permits an increase to 30 meters in length. In future, optical devices built into the connectors will adapt to optical fiber. Maximum data rate 20Gb/s, or more than 8 times as fast as Firewire.

LVDS – low voltage digital signaling, balanced unshielded pair that references a current through the wires rather than a specific voltage across them. Constant 3.5mA current in the loop; direction changes to indicate 1 or 0. With a typical 100Ω resistor across the receive end, this translates to a voltage of about 350mV.

CAN BUS – designed for automotive systems. Does not require a host computer; different nodes have different levels of priority and they automatically sort out bus clashes. Four wires including data pair and +12V and ground wires. Often shielded because of the electrically noisy automobile environment.

QWERTY KEYBOARD – the standard English-language keyboard layout. QWERTY refers to the first six keys in the upper row.

QUADRATURE – in a digital context, two signals or sensors spaced 90 degrees apart, so whenever one is switching, the other is holding. Two analog sine waves of the same frequency but 90° apart in phase are also said to be in quadrature.

CHAPTER 7 – MIXED DIGITAL / ANALOG

DIGITIZATION – converting an analog signal to a digital signal

QUANTIZATION – assigning a number value to an analog quantity. It is usually implied that the number is binary.
and the quantized number

QUANTIZATION ERROR – the difference between the actual analog signal voltage and the quantized number

Review page 18

SAMPLE – a “snapshot” of a signal voltage at a particular instant

BIT DEPTH – the number of bits in a sample; word length. The more bits, the closer the quantization can be to the analog voltage and the smaller the quantization error.

SAMPLING RATE - number of snapshot samples per second

NYQUIST THEOREM – a rule that says a continuing, repeating waveform can be sampled and then perfectly reconstructed if the waveform's highest frequency component does not exceed half the sampling frequency. A system that obeys this rule conforms to the NYQUIST CRITERIA.

ALIASING – what happens when we try to sample a frequency that's higher than half the sampling rate. The reconstructed waveform that comes out will be a different frequency than what went in. Subjectively, this is usually a very obnoxious form of distortion.

A-D Converter – Analog to Digital Converter, a circuit that converts an analog signal to a series of digital, usually binary, samples.

D-A Converter – Digital to Analog Converter, a circuit that takes a series of digital samples and reconstructs the original analog signal

MONOTONICITY – a converter in which increasing binary numbers always represent increasing voltage, step by step. If, at any point in the conversion characteristic, the circuit takes a backward step (an increase in the number corresponds to a decrease in voltage), the converter is NONMONOTONIC.

COUNTING CONVERTER – an A-D that converts an analog voltage to a binary number by counting up, one step at a time – 0, 1, 10, 11 etc – until a timed ramp or D-A output equals the input voltage. Simple, monotonic, slow.

SAR – Successive Approximation Register, an A-D that makes a series of guesses about the input voltage, each successive guess homing in on the correct value

FLASH – An A-D converter that uses many comparators connected to different points in a resistive divider string. A-D conversion is very fast and asynchronous, i.e. not tied to a clock. A very complicated circuit but well within the capabilities of modern IC manufacturing. (Not to be confused with Adobe Flash, the graphics program, or flash memory.)

DELTA MODULATION – digital representation of an analog signal where a binary value represents not voltage, but the change in voltage from the previous sample.

1-BIT CONVERTER – a converter based on delta modulation, where the output of each sampling operation is a single 1 or 0 depending on whether the sample needs to go up or down to track the analog voltage. 1-bit conversion is either very crude, or requires a very large number of samples per second to accurately represent a changing signal.

Review page 19

DECIMATION – converting a large number of 1-bit samples to a much smaller number of, say, 16-bit samples.

LINEAR QUANTIZING – the most common type of quantizing, where every binary increment represents the same size analog voltage increment

NONLINEAR QUANTIZING – a type of quantizing where the significance of a binary increment depends on the relative size of the analog voltage. When the voltage is small, the voltage increments represented by binary increments are small; when they're large the voltage increments represented by binary increments are large. This allows a signal that varies greatly in level to be quantized with reasonable accuracy at all levels.

DSP – Digital Signal Processing, making various modifications of an input signal by performing mathematical operations on the binary samples.

FIR – Finite Impulse Response filter, also called a TRANSVERSAL filter. Several slightly delayed copies of a signal are made, which are then combined in various ways with the undelayed signal to achieve a desired filter response.

COMB FILTER – a particular kind of FIR filter with deep response notches spaced evenly across a specified spectrum

PACKET – a chunk of data surrounded by routing instructions and error correction bits. The routing instructions allow a nearly unlimited number of unrelated packets to be sent over the same data channel, e.g. the internet, with each winding up where it's supposed to go. The error correction bits verify that the data has arrived in good shape, or if it hasn't, to request that it be sent over again.

LATENCY – delay between input and output. In a FIR filter, latency arises from the need to have several delayed copies of the signal. In some other systems, notably cellphones, latency is caused by the time needed to disassemble the analog signal into data packets, process the packets along with packets from many other signals, and then reassemble the packets into an analog signal.

DATA COMPRESSION – reducing the number of bits necessary to transmit information for the sake of efficient use of limited-capacity communication channels

MPEG – Motion Picture Experts Group, the industry association that established many features of the communication and compression standards in use

LOSSLESS COMPRESSION – data compression methods that do not permanently lose information, permitting the original signal to be perfectly recreated

LOSSY COMPRESSION – data compression methods that permanently lose some of the signal information, so it cannot be fully reconstructed. A common example is a video system that records and transmits color information with less resolution than brightness information.

Review page 20

INTERPOLATION – an attempt at recreating part of a missing waveform by creating a bridge between the previous and succeeding voltages.

Y, Cr, Cb – the three components that often compose a video picture: Y = brightness, Cr = red difference, Cb = blue difference (difference between the color and the Y brightness). Encoding video this way has two advantages: first, it can be shown on a monochrome (black and white) monitor, which only "sees" Y; second, the Cr and Cb signals can be transmitted at a lower data rate, thus reducing needed bandwidth, because the eye cannot pick out fine color details as it can with fine brightness details.

WAVEFORM GENERATION BY LOOKUP TABLE – creating a waveform by repeatedly stepping through a lookup table that holds the proper values.

WAVEFORM GENERATION BY COMPUTATION – creating a waveform by repeatedly performing a math operation in real time.

PSEUDORANDOM NOISE – a repeating binary pattern that exhibits the most important characteristics of nonrepeating, random noise. It is usually generated by waveform computation in real time by a comparatively simple circuit, or the circuit's equivalent in MCU firmware.

CHAPTER 8 – MACHINE CONTROL

CNC – Computer Numeric Control, machine control using binary (usually) numbers supplied by a computer

BINARY CONTROL – on / off or two level control, such as turning a motor on or off.

ANALOG CONTROL – multilevel control, usually a continuous range between on and off, analog values most often supplied by a binary computer an a digital-to-analog conversion circuit

OPEN LOOP – one-way communication from computer to machine. The computer issues instructions and the machine follows them. In data terms, a simplex system.

CLOSED LOOP – two-way communication from computer to machine and machine to computer. One advantage is that the computer can watch the machine to see if it is performing correctly, and issue corrections if necessary. In data terms, a full duplex or half duplex system.

SERVO – a motor or machine whose position or speed is governed by analog signals from a computer

HARDENED CIRCUITRY – circuitry furnished with input and output protection so as to be relatively immune to voltage transients, electromagnetic pulses etc.

Review page 21

HOT CHASSIS – what results when the internal ground and chassis of a unit are electrically connected to one side of the AC power line. Very dangerous, especially if the unit is connected to other units, which will also become hot.

DIN RAIL – a standardized mounting relay for electronic modules.

BRICK – an informal term for a module that would be installed on a DIN rail.

PHOENIX CONNECTOR – a wire termination method where the wire is inserted in the connector, then a screw is turned to tighten a clamp on the connector.

PLC – Programmable Logic Controller, a small computer furnished with the proper input and output circuitry to interface with typical machines

X, Y – in this context, not axes on a graph. In a PLC, inputs are X_0, X_1, X_2 etc and outputs are Y_0, Y_1, Y_2 etc. Note that counting starts from zero.

LADDER LOGIC – a visual way of assembling a program for a PLC, using symbols to represent X inputs and Y outputs. The resulting diagram often looks like the rungs of a ladder.

FUSED CONTACTS – relay or switch contacts that have melted together to form a permanently closed junction. When this happens, the machine that the relay controls cannot be turned off.

PULSE TRANSFORMER – a transformer designed to transfer short-duration pulses from primary to secondary winding instead of continuous sine waves.

ZERO CROSSING SWITCH – a circuit that emulates a switch which only closes and opens during the AC sine wave's zero crossings. The purpose is to eliminate turn-on and turn-off spikes and the very high surge currents that occur when contacts are closed at the peak of a wave

TRANSISTOR DISSIPATION – power that goes into a transistor as electricity and comes out as heat. When there is current through the transistor and voltage across it at the same time, the result is power that is dissipated by the transistor as if it were a resistor.

SOA – Safe Operating Area, a chart showing the limits of the voltage, current and power that a transistor can safely handle

DERATING - “marking down” from a transistor's published maximum ratings, which must be done when the transistor will be operating hotter than room temperature.

COMPARATOR INPUT – in a machine context, an analog input to an MCU or PLC that performs a comparator function, switching when an input signal reaches a preset value.

Review page 22

PWM OUTPUT – in a machine context, a method of converting an MCU's or PLC's digital output to an analog value via a variable duty cycle rectangle wave.

VFD – Variable Frequency Drive, a unit that generates a high-power AC sine wave that can be made to vary in frequency to control a large motor

WIRING REQUIREMENTS – actual power to motors etc is usually provided by electricians. The control tech's job is to make sure the control circuitry works correctly and is safe. Among other things, this involves:

- making sure there is no significant interference pickup or radiation. This involves shielding and balancing if necessary (depending on the application), watching grounds, and perhaps filtering.
- making sure the control wiring meets code. This involves proper wire gauges and insulation and often conduit.
- making sure there is no electrical connection between control circuitry people could touch and the AC power line...in other words, avoiding a hot chassis situation.

REED SWITCH – a pair of magnetically activated switch contacts inside a glass tube. A good, reliable switching method where the voltage and current across the switch are very low.

HALL EFFECT SWITCH – a solid state equivalent to the reed switch, which turns on or off in response to the strength and orientation of an external magnetic switches. Linear Hall components are also available, whose outputs change linearly with field strength.

KEYBOARD SWITCH – switch contacts in a computer keyboard, usually made of conductive rubber. They cannot handle high current and their closed resistance is nowhere near zero. To be used only in very low current circuits.

PHOTORESISTIVE CELL – a resistor whose value is inversely proportional to external illumination. Usually made of Cadmium Sulfide.

PHOTOTRANSISTOR – a bipolar transistor whose collector conductivity is proportional to illumination. Usually designed into a circuit where the available collector current is low, so the transistor will saturate in the presence of light. Photo-FETs are often available, though rare, but they are commonly used in paired LED / FET optical couplers.

PHOTODARLINGTON – a Darlington transistor pair where the low-powered transistor is photosensitive. A photodarlington's conductance is much higher than a phototransistor's, but it reacts to changes in light more slowly.

THERMISTOR – a variable-resistance element whose resistance depends on temperature. With **POSITIVE COEFFICIENT** thermistors, when temperature rises resistance rises. With **NEGATIVE COEFFICIENT** thermistors, when temperature rises resistance falls. The change in resistance can be made to occur suddenly at a particular temperature, e.g. a self-resetting circuit breaker.

Review page 23

CALIBRATION – creating a “chart” in memory of a sensor's nonlinearities, allowing the computer to compensate for them.

RELATIVE POSITION SENSOR – a position sensor that tells the computer when something has moved and how far it has moved

ABSOLUTE POSITION SENSOR – a position sensor that tells the computer explicitly where something is

INDEXING – computing the current position from a known start position and a known amount of movement away from that start; i.e. a way to use a relative position sensor to compute absolute position

TACHOMETER – a rotary sensor that puts out a certain number of pulses per revolution

STEPPER MOTOR – a motor that rotates in steps, one step at a time. Often takes the form of a motor with two coils in quadrature, where two square waves 90° out of phase determine speed and direction of rotation

MICROSTEPPING – tricking a stepper into taking positions between steps by partly activating one or more coils. For example, in a 2-coil 6V motor, putting 4V on one coil and 2V on the other will position the rotor between steps

GRAY CODE – a binary coding system used in absolute position sensing systems that eliminates the uncertainty between some binary increments

STABILITY AND DAMPING – issues relating to servo systems with corrective feedback, and the stability problems that can arise

OVERSHOOT – when a machine or mechanical function goes past its target position due to momentum

CRITICAL DAMPING – a servo loop adjusted so the machine responds to commands as fast as possible without serious overshoot or instability

PID – Proportional, Integral, Differential control, a control technique that combines in-the-moment, forward-looking and backward-looking factors to provide a faster and at the same time more stable servo loop

SET POINTS – high and low limits that turn a mechanical function on and off. Used in applications where very precise control would be undesirable because it would involve a lot of power expended for little benefit.

FLYBACK SUPPRESSION IN A DC CIRCUIT – usually accomplished with a snubber diode.

Review page 24

FLYBACK SUPPRESSION IN AN AC CIRCUIT – accomplished via back-to-back zeners with a comparatively high breakdown voltage, or via a series resistor / capacitor string connected across the switch contacts

CHAPTER 9 – DISPLAYS AND PRINTERS

2-DIMENSIONAL DISPLAYS – almost all displays are flat, 2-dimensional displays with little or no actual depth.

IMITATING 3D – turning squares into trapezoids, and similar techniques, so as to trick the mind into seeing depth. Mother Nature does this as well; if you take a square object and turn it at an angle, what you see with either eye is a trapezoid.

IMITATING STEREOSCOPY – creating two 2-dimensional images with slightly different viewpoints, and routing each to the appropriate eye

1-DIMENSIONAL DATA STREAM – for transmission and storage, a 2-dimensional image is broken down into essentially 1-dimensional lines that can be handled as a data stream

RASTER – a picture made of many horizontal lines, as in the days of CRTs when an electron beam would actually scan across the stream

TIME DIVISION MULTIPLEXING – sending information to many places via one wire by allotting a certain time period to each

VIDEO – electronic display of moving images

FRAME – one complete raster scan, one set of all the lines it takes to make an image

FRAME RATE – the number of frames per second

HORIZONTAL SCANNING FREQUENCY – the number of lines scanned per second. In digital displays there is no scanning electron beam; instead, “scanning” means sequentially sending information to the pixels in a display via time division multiplexing.

COLOR PALETTE – all the colors that can be reproduced by a video system

LUMINANCE – the bright-and-dark information about a particular place in an image; i.e. the black-and-white part.

CHROMINANCE – the color information about a place in the image; e.g. red vs. blue vs. orange etc

SATURATION – how strong the color is in a certain place; e.g. bright red vs. gray-red vs. gray (zero saturation)

Review page 25

COMPOSITE VIDEO – a video system where all the brightness, color and synchronization information is combined into a single waveform

NTSC – the traditional broadcast television system, using composite video

S-VIDEO – similar to composite video except that the color information is removed from the composite video and relayed over a second wire. What remains on the first wire is essentially black-and-white TV.

LINES – the number of horizontal or vertical lines that can be resolved on the screen; that is, resolution. In traditional video, the vertical resolution is set by the NTSC standard to around 480 lines. The horizontal resolution depends on the quality of the system but can be as good as 500-600 lines.

PIXELS – the digital equivalent to lines; how many dots the picture is made op of vertically and horizontally. More pixels allow us to display finer detail.

VGA – the analog wiring standard developed specifically to connect monitors to computers. Sends red, green, blue, horizontal sync and vertical sync as five separate colors. There are higher-resolution standards like XGA that use the same wiring format. Signals are single-ended; for each there is a wire inside a shield.

DVI – the most widely used digital standard for connecting monitors to computers. Signals sent differentially, over balanced pairs.

HDMI – similar to DVI, intended for displayport. Signals send differentially, over balanced pairs.

DISPLAYPORT – a video standard that treats the components of an image as data packets, offering great flexibility in what is sent and how. Usable with computer equipment that has THUNDERBOLT capabilities.

LCD – Liquid Crystal Display

POLARIZATION – filtering light so that only waves that are oriented in a certain direction gets through. An LCD film between two polarizing layers twists light by varying degrees to make it pass or not pass the second filter.

PLASMA – gas molecules in an excited state, where some electrons have absorbed extra energy and temporarily in a higher than normal orbit. When they fall back to their regular orbit, they emit light.

LENTICULAR LENS – a lens made of tiny, thin, long ridges in transparent material. A lenticular lens in an LCD bends light toward where people are probably sitting.

DICHROIC MIRROR – a mirror that reflects some light wavelengths while passing others through, like glass, thus splitting white light into two color ranges. These ranges may then be further split.

Review page 26

FIELD SEQUENTIAL – creating color images by showing a complete red image, then a complete green image and so on, changing the color with each field.

COLOR WHEEL – a wheel placed between the light source and LCD or DLP panel to “dye” the light into the proper colors in sequence. A widely used way to achieve a field sequential display.

SEGMENTED DISPLAY – a display that is not a video picture. Often LCD, but the liquid crystals do not make up raster pixels, they are meaningful shapes in themselves.

ELECTROLUMINESCENT PANEL – a phosphor-based panel where the phosphor is directly excited by an AC electric field. The typical excitation is around 120V at 1kHz.

E-PAPER – a display that resembles paper in that it does not glow, but is lit by ambient light the way regular paper is.

THERMAL PRINTER – a printer that makes images on special paper via heat

INKJET PRINTER – a printer that makes an image by spraying tiny drops of ink on paper

THERMAL INKJET PRINT HEAD – makes ink move from print head to paper by vaporizing it. Not to be confused with the type of thermal printer that makes images by heating treated paper.

PIEZOELECTRIC PRINT HEAD – uses the movement of a piezoelectric material to fling ink droplets onto the paper

CIJ – Continuous In Jet, a printer where ink flows continuously but is electrostatically deflected either onto the paper or to a recycling collector.

XEROGRAPHY – a copying technology that uses an electrically charged surface, and photoresistive effects to drain off the charge in certain areas, to control which areas attract powdered ink (called “toner”).

LASER PRINTER – uses a version of Xerography where the pattern of charged / discharged areas is created by a scanning laser

ADDITIVE PROCESS – red, green and blue light in varying amounts are summed to create a color. Full-strength red + green + blue light = white. The colors are abbreviated RGB.

SUBTRACTIVE PROCESS – cyan, yellow and magenta ink in varying amounts are mixed to create color. Cyan absorbs red light, yellow absorbs blue light and magenta absorbs green light. In theory, full-strength cyan + yellow + magenta ink = black. In practice, a fourth ink, actual black, is added to get true dark blacks. The colors are abbreviated CYMK, where “K” refers to black because it's the image layer that the others are keyed to.

Review page 27

CONVERGENCE – in a color display, adjusting the three color images so they fall exactly on top of each other. Only an issue when the red, green and blue images are made by separate displays as in 3-DLP and 3-LCD projectors. This problem is handled at the factory.

REGISTRATION – in a color printer, adjusting the four color images so they fall exactly on top of each other. The black image (K) is taken as the key and the cyan, yellow and magenta images are adjusted to match it.

RESOLUTION – the number of discrete picture details a display or printer is capable of resolving.

- In CRTs, resolution is measured in the number of “lines” that can be resolved from the left side of the screen to the right side and from top to bottom. Horizontal and vertical resolution are often different.
- In LCD and other digital technologies, resolution is measured in picture elements or “pixels,” which are the digital equivalent of lines. Pixels are usually square, so horizontal and vertical resolution is essentially the same. It may not seem like it, because the numbers are different, but that's because the picture is wider than it is high.
- In printers, resolution is not measured in terms of picture elements across the page, but picture elements per inch, as in dots per inch or DPI.

CHAPTER 10 – DIGITAL SERVICING TECHNIQUES

COMMON CAUSES OF DATA DEGRADATION

- noise – random energy that may swamp a weak signal. Reduce the noise, if possible, or move the equipment away from the noise source. In the case of a signal that's so weak normal environmental noise competes with it, the only remedy is to strengthen the signal.
- transients – short, random bursts of energy. The remedy is the same as for noise.
- high frequency attenuation and delay – usually due to long cables. Replace with shorter cables if possible, or cables with lower loss. Insert a booster amplifier with equalization somewhere along the cable's length.
- improper termination – if the sending and receiving units have a different impedance than the cable, there will be signal reflections at the end. It is sometimes necessary to insert a termination network that loses level, but in many systems there is level to spare.
- Ringing – a resonance at some frequency. This problem usually occurs inside a piece of equipment. It is unlikely to occur with cables that are properly terminated.
- Crosstalk – data leakage from one place to another. It is sometimes necessary to physically separate the parts of a system to prevent crosstalk. An interface that uses balanced pairs, such as Thunderbolt, seldom has crosstalk issues.
- Ground loops – occurs when widely separated grounds cause 60Hz hum and other kinds of interference to be induced in a system. The best way to become familiar with this problem is to study EM3, which has a detailed discussion.

Review page 28

- Ground bump – when a particular ground, such as Earth ground, has a momentary voltage difference from the system ground, or when there is a discrepancy between grounds within a system.
- Rail bump – when the voltage on the DC rail momentarily changes with reference to ground.
- Skew – when the timing of a data stream is advanced or delayed relative to the clock.
- Jitter – when the timing of a data stream is erratic.
- ESD – electrostatic discharge which, at minimum, can cause data errors and at maximum can damage equipment. The best defense is to make sure equipment is Earth-grounded, though that may cause ground loop problems. Drawing a diagram showing every ground in the system often helps to improve the grounding scheme.

TEST EQUIPMENT

- logic pulser – generates “0” and “1” pulses that may be fed into a circuit to force level transitions.
- logic probe – uses a LED to indicate logic state / activity at a selected point in a circuit.
- current probe – detects the magnetic field created by data flowing in a conductor.
- logic analyzer – a special kind of oscilloscope with many probes, permitting the user to monitor many points in a circuit.
- spectrum analyzer – a unit that displays spectral energy at each frequency over a selected span of frequencies.

HARMONIC ANALYSIS – one use of a spectrum analyzer, displaying the relative strengths of the various harmonics in a waveform

HARMONIC DISTORTION – distortion of a waveform, causing harmonics to be produced

INTERMODULATION DISTORTION – what happens when two waveforms modulate each other, causing new frequencies to be produced

FOURIER ANALYSIS – a real-time spectrum analysis method where frequency energy in the signal is sorted into “bins”

SAMPLING NOISE GAIN – causes the noise floor shown on a spectrum analyzer to be much lower than the actual noise floor

OSCILLOSCOPE – creates a real-time graph of voltage vs. time. A dual-trace oscilloscope can be used to compare timing and waveshape etc of two waveforms in real time.

ATTENUATOR PROBE – a probe with a built in attenuator, usually a 10:1 ratio. In most cases the attenuator can be switched in or out. Most oscilloscope probes are attenuator probes.

ACTIVE PROBE – a probe with a built-in amplifier, used to isolate the circuit being tested from the resistive and capacitive loading of the probe cable and oscilloscope input.

Review page 29

TIMING DIAGRAM – a voltage-vs.-time graph of several waveforms, showing the timing differences among the rise and fall transitions among waveforms.

CURSOR – a vertical or horizontal line that can be moved across a scope face to pinpoint a place on the displayed waveform. Most modern scopes can measure voltage at any selected point in a wave by moving the cursor to that point.

CALIBRATION – adjusting test gear to a known accurate reference standard.

VECTOR ANALYZER – a unit that shows both voltage and phase of a signal.

DATA ANALYZER – a unit that combines oscilloscope (voltage vs. time), spectrum analysis (voltage vs. frequency), vector analysis and data decoding and display.

DIGITAL WAVEFORM GENERATOR – a unit that generates an analog waveform digitally, e.g. by looking up values in a lookup table.

ARBITRARY WAVEFORM GENERATOR – a digital generator that can be set to generate any possible waveform.

BLOCK DIAGRAM – a diagram that shows the major functional parts of a system as blocks.

SCHEMATIC DIAGRAM – a diagram that shows actual parts and connections.

TEST POINT – a point in a circuit which the manufacturer specifies as a good place to attach an analyzer probe. This is usually an important measuring point and there may be a separate amplifier to isolate the point from the rest of the circuit to prevent loading.

BUILD-FOR TEST, SELF-TEST – describes a unit with some test functions built in.

DIAGNOSTIC SOFTWARE – a testing program supplied by the manufacturer of a piece of equipment, designed to run a sophisticated test of all its functions.

EQUALIZATION – deliberate alteration of the gain vs. frequency characteristic of a system. In digital systems, used to counteract the high-frequency attenuation and delay of long cables.

REPEATER – an amplifier connected into a signal or data line that amplifies and equalizes the signal. Used where cable attenuation is so great that the signal cannot make it from one end to the other without help along the way.

Review page 30

To a large extent, servicing techniques come down to precautions, reminders and tales from experience:

Don't always trust your test gear. When a measurement isn't what you expect, there may be something wrong with the unit under test – or with the way you're testing it. Double check to see that the settings are right.

The test gear may be affecting the measurement. Know how the test gear behaves and what its input impedance is, so you can know when it's loading down the circuit its' testing.

Watch out for grounds. If your test gear isn't grounded in the right place, you will see strange results.

Keep the limitations of your test gear in mind. For example, an oscilloscope's frequency response only goes so high. A multimeter's AC function is only accurate on 60Hz sine waves.

Most interconnection problems are analog. That is, resistive or capacitive loading, high frequency attenuation, ground loops.

Don't exceed the input voltage capability of your test equipment. This applies especially to active oscilloscope probes.

To find an intermittent function, tap sharply on various components with an insulated tool. It also helps to use freeze spray to change the temperature of various components, one after another.

Ask questions – when you arrive on a customer's site, ask questions about what has happened recently that might have caused the malfunction you're looking for.

Review page 31

EM4 CHARTS AND TABLES

BINARY NUMBERS

no. of bits	*no. of possible values*	*lowest binary value*	*highest binary value*	*highest decimal value*
1	2	0	1	1
2	4	00	11	3
3	8	000	111	7
4	16	0000	1111	15
5	32	00000	11111	31
6	64	000000	111111	63
7	128	0000000	1111111	127
8	256	00000000	11111111	255

1-BYTE NUMBERS

Decimal number	Hexadecimal number	Decimal number	Hexadecimal number
00	00	16	10
01	01	17	11
02	02	18	12
03	03	19	13
04	04	20	14
05	05	21	15
06	06	22	16
07	07	23	17
08	08	24	18
09	09	25	19
10	0A	26	1A
11	0B	27	1B
12	0C		
13	0D	all the way to...	
14	0E		
15	0F	255	FF

BINARY ARITHMETIC TABLES

0 + 0 = 0
1 + 0 or 0 + 1 = 1
1 + 1 = 0, carry 1

0 – 0 = 0
0 – 1 = 1, borrow 1

1 – 0 = 1
1 – 1 = 0

0 x 0 = 0
1 x 0 or 0 x 1 = 0
1 x 1 = 1

0 / 1 = 0
1 / 1 = 1

BOOLEAN MATH RULES

$(A \bullet B) = (B \bullet A)$ $(A \bullet B) \bullet C = A \bullet (B \bullet C)$

$(A + B) = (B + A)$ $(A + B) + C = A + (B + C)$

commutative rules — associative rules

$A \bullet A = A$ $A + A = A$ identity rules

$A \bullet \overline{A} = 0$ $A + \overline{A} = 1$ not-identity rules

$A \bullet (B \bullet C) = (A \bullet B) \bullet (A \bullet C)$

$A \bullet (B + C) = (A \bullet B) + (A \bullet C)$

distributive rules

Review page 32

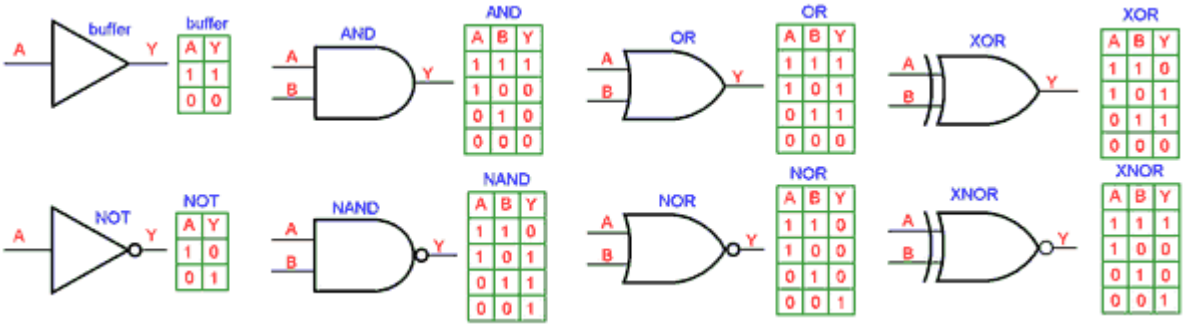
$$A + (A \bullet B) = A$$
$$A \bullet (A + B) = A$$

redundancy rules

$$(A \bullet B) + (A \bullet \overline{B}) = A$$
$$(A + B) \bullet (A + \overline{B}) = A$$

another redundancy

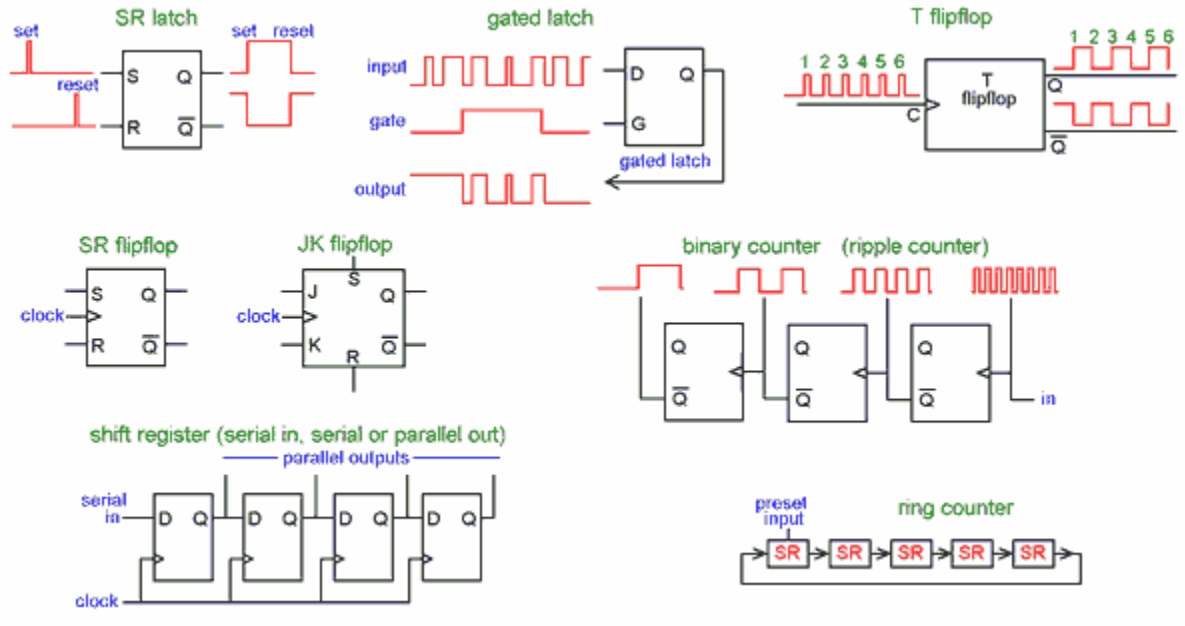
BOOLEAN GATES AND NOTATION



buffer $Y = A$ AND $Y = A \bullet B$ OR $Y = A + B$ XOR $Y = A \oplus B$

NOT $Y = \overline{A}$ NAND $Y = \overline{A \bullet B}$ NOR $Y = \overline{A + B}$ XNOR $Y = \overline{A \oplus B}$

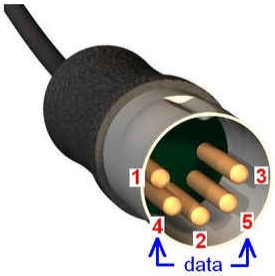
FLIPFLOPS



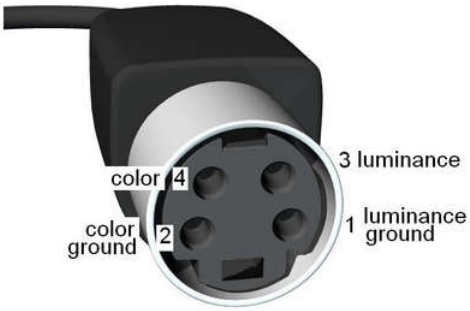
Review page 33

DATA LINK & VIDEO STANDARDS

MIDI



S-VIDEO



RS232

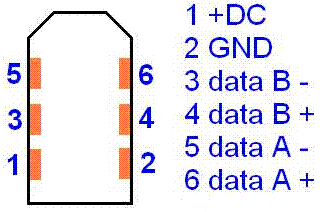
Name	Typical purpose	DB25	DB9	8P8C/RJ45
Data Terminal Ready	OOB control signal: Tells DCE that DTE is ready to be connected.	20	4	3
Data Carrier Detect	OOB control signal: Tells DTE that DCE is connected to telephone line.	8	1	2
Data Set Ready	OOB control signal: Tells DTE that DCE is ready to receive commands or data.	6	6	1
Ring Indicator	OOB control signal: Tells DTE that DCE has detected a ring signal on the telephone line.	22	9	
Request To Send	OOB control signal: Tells DCE to prepare to accept data from DTE.	4	7	8
Clear To Send	OOB control signal: Acknowledges RTS and allows DTE to transmit.	5	8	7
Transmitted Data	Data from computer to peripheral	2	3	6
Received Data	Data data from peripheral to computer	3	2	5
Common Ground		7	5	4
Protective Ground		1	—	—

ETHERNET CONNECTIONS

MIDI

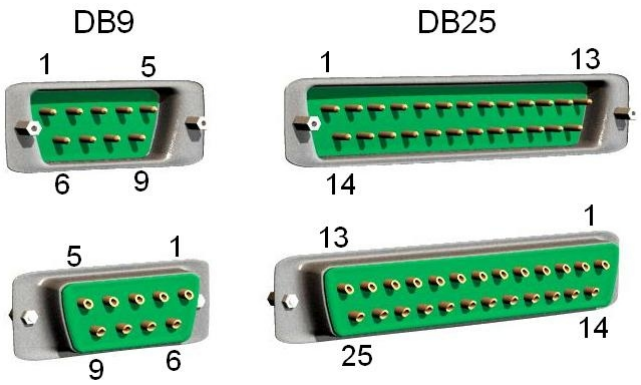
S-VIDEO

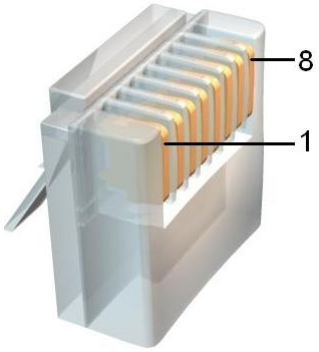
IEEE 1394 (CABLE END)



8P8C / RJ8

Review page 34

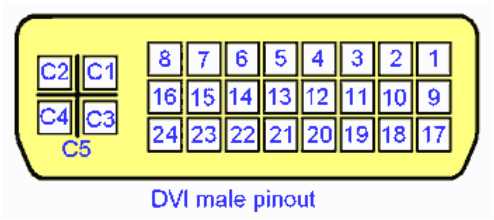




VGA
(15-PIN MINI D CONNECTOR)

1	Red video
2	Green video
3	Blue video
4	(reserved)
5	Horizontal sync shield/ground
6	Red shield/ground
7	Green shield/ground
8	Blue shield/ground
9	+5VDC
10	Vertical sync shield/ground
11	(reserved)
12	I^2C data
13	Horizontal sync
14	Vertical sync
15	I^2C clock

DVI

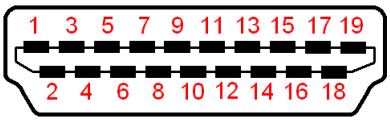


1	Data 2, Red 1-
2	Data 2, Red 1+
3	Data 2 & 4 shield
4	Data 4, Green 2-
5	Data 4, Green 2+
6	DDC clock
7	DDC data (I^2C)
8	Analog vertical sync
9	Data 1, Green 1-
10	Data 1, Green 1+
11	Data 1 & 3 shield
12	Data 3, Blue 2-
13	Data 3, Blue 2+
14	+5V
15	Ground, return for analog sync and 5V
16	Hot plug detect
17	Data 0, Blue 1-
18	Data 0, Blue 1+
19	Data 0 & 5 shield
20	Data 5, Red 2 -
21	Data 5, Red 2+
22	Data clock shield
23	Data clock +
24	Data clock -
C1	Analog red
C2	Analog green
C3	Analog blue
C4	Analog horizontal sync
C5	Analog ground

Review page 35

HDMI

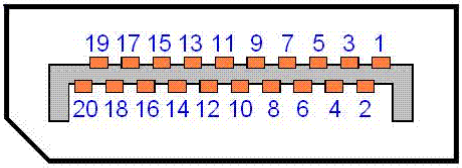
1	Data 2+
2	**Data 2 shield**
3	**Data 2-**
4	**Data 1+**
5	**Data 1 shield**
6	**Data 1-**
7	**Data 0+**
8	**Data 0 shield**
9	**Data 0-**
10	**Data clock +**
11	**Data clock shield**
12	**Data clock -**
13	**CEC**
14	**HEC data**
15	**I²C clock**
16	**I²C data**
17	**I²C ground**
18	**+5V, 50mA max**
19	**Hot plug detect**



CABLE END

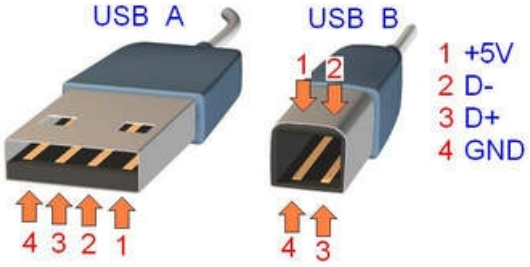
DISPLAYPORT

PIN	SIGNAL	PIN	SIGNAL
1	**lane 0 +**	**11**	**lane 3 gnd**
2	**lane 0 gnd**	**12**	**lane 3 -**
3	**lane 0 -**	**13**	**config 1 (gnd)**
4	**lane 1 +**	**14**	**config 2 (gnd)**
5	**lane 0 gnd**	**15**	**aux chan +**
6	**lane 1 -**	**16**	**aux gnd**
7	**lane 2 +**	**17**	**aux chan -**
8	**lane 2 gnd**	**18**	**hot plug detect**
9	**lane 2 -**	**19**	**power return**
10	**lane 3 +**	**20**	**power 3.3V 0.5A**

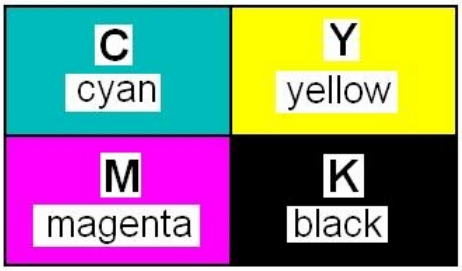


equipment end pinout

USB



PRINTER COLORS



Review page 36

GRAY CODE

```
0000 0                    1100 8
0001 1                    1101 9
0011 2                    1111 10
0010 3                    1110 11
0110 4                    1010 12
0111 5                    1011 13
0101 6                    1001 14
0100 7                    1000 15
```

ASCII CHART

one 3-bit word and one 4-bit word

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	'	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	.	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	-	DEL

ASCII control characters

NUL	null
SOH	start of heading
STX	start of text
ETX	end of text
EOT	end of transmission
ENQ	enquiry
ACK	acknowledge
BEL	ring bell
BS	backspace
HT	horizontal tab
LF	line feed (new line)
VT	vertical tab
FF	form feed (new page
CR	carriage return
SO	shift out
SI	shift in

DLE	data link escape
DC1	device control 1
DC2	device control 2
DC3	device control 3
DC4	device control 4
NAK	negative acknowledge
SYN	synchronous idle
ETB	end of transmission block
CAN	cancel
EM	end of medium
SUB	substitute
ESC	escape
FS	file separator
GS	group separator
RS	record separator
US	unit separator
DEL	delete

note – ASCII was developed in the days of printers that used mechanical printing means like a typewriter, which is why the commands are so oriented toward printing.

End of review

EM4 PRACTICE EXAMS

Print these out and put them in a folder. Take the exams one chapter at a time, after you've read through the chapter and the review enough to be confident that you know the material. Answers for all exams are listed together, after the last exam.

Note that some chapters have a *lot* of questions. I've heard different opinions about this; some people are afraid this will be intimidating to the student. My feeling is that there is no reason to arbitrarily limit the number of questions, since this is a practice exam and you're free to not answer them all if you don't feel like it. But I'd suggest doing them all, and also going back and reviewing until your knowledge is rock solid. Besides, you've paid for this study guide, might as well get your money's worth.

Except where noted, using a calculator and the charts and memory aids printed in the review is "legal." You'll need paper and pencil to work out the Karnaugh map question.

CHAPTER 1 – DIGITAL BASICS

1. Digital signal / data takes the form of:
   A. an infinite range of voltage levels
   B. specific, discrete voltage levels
   C. sine waves
   D. sawtooth waves

2. Most of the time, the number of "states" or voltage levels used by digital circuitry is:
   A. 2
   B. 3
   C. 5
   D. 7

3. The word that describes the number of states used by most digital circuitry is:
   A. quadrature
   B. binary
   C. decimal
   D. bipolar

4. "Pull down" in the context of binary logic means:
   A. loading the +DC supply voltage until it sags
   B. discharging a capacitor to ground
   C. a change from logic 1 to logic 0
   D. using a resistor or transistor to pull the voltage at a point in the circuit to ground

5. "TTL" means:
   A. transistor-transistor loading
   B. transformer-transistor levels
   C. transistor translation logic
   D. transistor-transistor logic

6. The standard DC supply voltage for TTL is:
   A. +1V
   B. -5V
   C. +15V
   D. +5V

7. "MOS" means:
   A. metal open source
   B. material oxide semiconductor
   C. metal oxide semiconductor
   D. metal octal semiconductor

8. "CMOS" means:
   A. capacitive MOS
   B. combined MOS
   C. complementary MOS
   D. connected MOS

9. The +DC supply voltage for a 4016 CMOS IC is:
   A. 8V to 15V
   B. 3V to 15V
   C. 3V to 8V
   D. -5V

10. A 74HCT14 is:
   A. a lower-power, vaster version of TTL
   B. a standard-CMOS version of a TTL IC
   C. an IC manufactured in the 17th week of 1974
   D. a CMOS IC that can work with TTL circuitry

11. The voltage levels or states used in binary circuitry are commonly called:
   A. high and low
   B. in and out
   C. 1 and 0
   D. A and C above

12. "Pinout" means:
   A. how long the IC pins are
   B. which internal circuit points are connected to which pins of an IC
   C. which pin must be grounded first when an IC is plugged in
   D. how much current the IC can supply to its output terminals

13. “Active low” means
    A. the low-voltage state represents “1”
    B. the low-voltage state represents “0”
    C. the circuit draws most current when in the “0” state
    D. 1-0 transitions happen faster than 0-1

14. An “undefined” state is a voltage that:
    A. is poorly regulated
    B. is higher than the IC's proper power supply voltage range
    C. oscillates regularly between 1 and 0
    D. is not high enough to be “1” or low enough to be “0”

15. For TTL circuitry, an undefined voltage is:
    A. between 0.8V and 2V
    B. between 1.5V and 3.5V
    C. between 3.5V and 5V
    D. between 0V and 2V

16. For CMOS running on a 5V supply, an undefined voltage is:
    A. between 0.8V and 2V
    B. between 1.5V and 3.5V
    C. between 3.5V and 5V
    D. between 0V and 2V

17. The input of a CMOS circuit is essentially:
    A. a virtual ground
    B. an open circuit
    C. an inductance
    D. a fuse

18. A CMOS input should never be:
    A. left floating
    B. tied to ground
    C. tied to +DC
    D. tied to the output of its own gate

19. Shoot-through occurs:
    A. when both transistors of a complementary pair are “on”
    B. during 0-1 or 1-0 transitions
    C. if an input is left floating
    D. all of the above

20. ESD is the abbreviation of:
    A. electromagnetic damage
    B. electrostatic discharge
    C. electrostatic danger
D. electronic destruction

21. ESD shunt diodes:
    A. drain ESD discharges off to the + rail or ground
    B. reroute ESD discharges to the gate's output
    C. reroute ESD charges to a filter capacitor
    D. short out the power supply during ESD events

22. A build-out resistor:
    A. is in series with a circuit input or output
    B. is in series with the power supply rail
    C. is connected between an input and ground
    D. is connected across the power supply

23. A 74LS14 IC is:
    A. A CMOS equivalent of TTL
    B. slower than standard TTL
    C. low-power Schottky TTL
    D. this is a date code

CHAPTER 2 – BINARY ARITHMETIC

1. The set of numbers that a number system is based on is called (two correct answers):
    A. base
    B. decimal
    C. span
    D. radix

2. Binary 1000 is _____ in decimal.
    A. 4
    B. 8
    C. 15
    D. 16

3. Binary 1001010 is _____ in decimal.
    A. 7
    B. 37
    C. 74
    D. 121

Do no. 4 and 5 manually – do not use a calculator.

4. Binary 1001011 + 0010111 =
    A. 1011101
    B. 1011111
    C. 1100010
    D. 0110100

5. Binary 1001011 x 0010111 =
    A. 00110111011
    B. 11010111101
    C. 10111001111
    D. 10100110111

6. Which addition rule is not correct?
   A. 0 + 0 = 0
   B. 0 + 1 = 1
   C. 1 + 0 = 0
   D. 1 + 1 = 0, carry 1

7. Which multiplication rule is not correct?
   A. 0 x 0 = 0
   B. 1 x 0 = 1
   C. 0 x 1 = 0
   D. 1 x 1 = 1

8. Which subtraction rule is not correct?
   A. 0 – 0 = 0
   B. 0 – 1 = 1, carry 1
   C. 1 – 0 = 1
   D. 1 – 1 = 0

9. Which division rule is not correct? (two answers)
   A. 0 / 0 = 0
   B. 1 / 0 = 1
   C. 0 / 1 = 0
   D. 1 / 1 = 1

10. Octal notation uses a set of ___ numbers, which are:
   A. eight, 0 thru 7
   B. eight, 1 thru 8
   C. eight, 00000000 thru 11111111
   D. sixteen, 0 thru F

11. Octal 5 is binary _____.
   A. 011
   B. 101
   C. 110
   D. 111

12. Hexadecimal uses a set of ____ numbers, which are:
   A. sixteen, 1 thru F
   B. sixteen, 0 thru F
   C. sixteen, 0 thru 15
   D. sixty, 0 thru 59

13. Hex D is binary:
   A. 0101
   B. 0111
   C. 1101
   D. 1110

14. Hex D is decimal:
   A. 11
   B. 40
   C. 17
   D. 13

15. A bit is:
   A. one binary digit
   B. always 1
   C. always 0
   D. a carry operation

16. A Byte is:
   A. two bits
   B. four bits
   C. eight bits
   D. sixteen bits

17. BCD (binary coded decimal) is:
   A. a way of using decimal numbers to count in binary
   B. a way of using binary to count in base 10
   C. an obsolete counting system
   D. used for numbers larger than $F

18. The maximum binary value for one decimal digit in BCD is:
   A. 1001
   B. 0111
   C. 1011
   D. 1110

19. EBCDIC is:
   A. an 8-bit alphanumeric code invented by IBM
   B. a 16-bit alphanumeric code invented by Intel
   C. a different way of counting in binary
   D. an Internet protocol

20. ASCII is:
   A. a machine control language
   B. a 7-bit alphanumeric code invented by ANSI
   C. a BCD calculation language
   D. an encryption code used by government

21. UTF-8 is:
   A. an international code unrelated to ASCII
   B. a secret code used by other countries
   C. a code designed to work like ASCII but faster
   D. an extension of ASCII to handle languages other than English

CHAPTER 3 – BOOLEAN LOGIC

1. Circuits that perform Boolean logic operations are called:
   A. amplifiers
   B. gates
   C. nodes
   D. terminals

2. Which of the following AND statements is wrong?
   A. A = 0 B = 0 Y = 0
   B. A = 0 B = 1 Y = 1
   C. A = 1 B = 0 Y = 0
   D. A = 1 B = 1 Y = 1

3. Which of the following OR statements is wrong?
   A. A = 0 B = 0 Y = 0
   B. A = 0 B = 1 Y = 1
   C. A = 1 B = 0 Y = 0
   D. A = 1 B = 1 Y = 1

4. The statement Y = A apples to which device?
   A. inverting buffer
   B. noninverting buffer
   C. NOR
   D. NAND

5. An inverting buffer is more commonly known as:
   A. NOT-AND
   B. NOT-OR
   C. NOT
   D. exclusive-OR

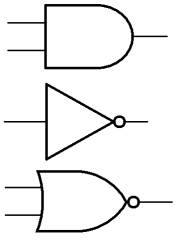
6. In text, the opposite of A may be written as:
   A. nA
   B. notA
   C. -A
   D. any of the above

7. In diagrams, the opposite of A is shown as:
   A. A with a line through it
   B. A with a line over it
   C. A inside a circle
   D. upside down A

8. An octal inverting buffer consists of:
   A. eight NOTs
   B. six NOTs
   C. an 8-input NOT
   D. eight buffers with an active-low enable

9. Which of the following NAND statements is wrong?
   A. A = 0 B = 0 Y = 1
   B. A = 0 B = 1 Y = 1
   C. A = 1 B = 0 Y = 1
   D. A = 1 B = 1 Y = 1

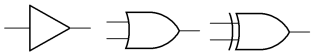
10. From top to bottom, the symbols to the right are:
   A. AND, NOT, NOR
   B. AND, NOT, OR
   C. NAND, BUFFER, NOR
   D. OR, NOT, NAND

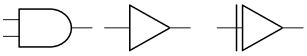


11. The three states of a tri-state output are:
   A. 0, 1, 1/2
   B. 0, 1, open
   C. 0, 1, short
   D. 0 out, 1 out, reverse (becomes input)

12. The diagram below where all symbols are correct is:

A.

B. 

C. 

D.

13. An open collector / drain:
   A. is another way of describing a tri-state output
   B. pulls up but not down
   C. pulls down but not up
   D. normally rests at 2.5V

14. The truth table to the right is a/an:
   A. XOR
   B. NAND
   C. NOR
   D. XNOR

A	B	Y
1	1	0
1	0	1
0	1	1
0	0	0

15. The expression A · B describes an:
   A. OR
   B. NAND
   C. AND
   D. XOR

16. The expression A + B describes an:
   A. AND
   B. OR
   C. XOR
   D. NAND

17. A NOR is:
   A. an OR followed by an AND
   B. an OR followed by a NOT
   C. an AND followed by a NOT
   D. an XOR followed by a NOT

18. (A + B) + C is the same as:
   A. (A · B) · C
   B. (A · B) / C
   C. (A + B) · C
   D. A + (B + C)

19. A + B is the same as:
    A. A · B
    B. B + A
    C. A + notB
    D. A · notB

20. A + notA =
    A. 1
    B. 0
    C. A
    D. this is an illegal operation

21. A · notA =
    A. this is an illegal operaton
    B. 1
    C. A
    D. 0

22. A + (A · B) =
    A. B
    B. A
    C. 2A + B
    D. 2A · B

23. A · (A + B) =
    A. 1
    B. A
    C. 0
    D. cannot be determined without more info

24. These equations are called:

$$\overline{A \bullet B} = \overline{A} + \overline{B}$$
$$\overline{A + B} = \overline{A} \bullet \overline{B}$$

    A. Karnaugh's theorems
    B. deMorgan's identities
    C. deMorgan's theorems
    D. the distributive principle

25. Use a Karnaugh map to find the Boolean equation that corresponds to this truth table.

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

    A. $B\overline{C} + A\overline{B}C = Y$
    B. $\overline{A}\overline{B}\overline{C} + \overline{A}B = Y$
    C. $\overline{B}\overline{C} + \overline{A}B = Y$
    D. $\overline{A}B + \overline{B}C = Y$

CHAPTER 4 – SEQUENTIAL LOGIC

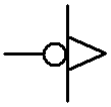
All questions refer to active-high logic except where noted.

1. To "flip" or "set" is to:
    A. change state from 1 to 0
    B. change state from 0 to 1.
    C. momentarily go high, then low again
    D. reverse state (from 0 to 1 or 1 to 0)

2. Three ways to describe a 1-to-0 transition are:
    A. flop, reset, clear
    B. flop, cancel, kill
    C. flop, invert, clear
    D. J, clear, reset

3. Operation of a synchronous circuit is governed by:
    A. a flipflop
    B. a ring counter
    C. a clock
    D. a watchdog timer

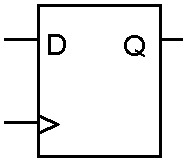
4. This symbol identifies a/an:



    A. inverting buffer
    B. positive-going edge triggered clock input
    C. negative-going edge triggered clock input
    D. enable input

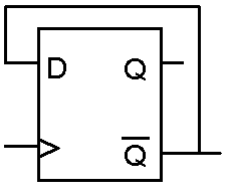
5. A latch:
    A. enables or disables an output
    B. starts or stops a ripple counter
    C. presets J and K inputs
    D. captures and holds an input state

6. This circuit is a/an:



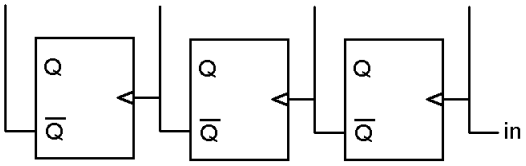
    A. gated latch
    B. clocked D latch
    C. data inverter
    D. SR flipflop

7. This circuit shows one way of making a/an:

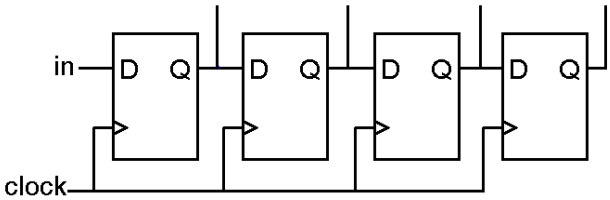


    A. XOR
    B. clocked latch
    C. JK flipflop
    D. T flipflop

8. T flifplops connected as shown below make a/an: (two correct answers)
   A. shift register
   B. binary counter
   C. ripple counter
   D. ring counter



9. The circuit below is a/an:
   A. serial-in, parallel-out shift register
   B. parallel-in, serial-out shift register
   C. binary counter
   D. ring counter



10. In a clocked SR flipflop, if S = 0 and R = 0, what happens to Q when the clock occurs?
   A. nothing (stays the same)
   B. Q changes state
   C. Q goes to 1 and back to 0
   D. Q flops or stays flopped

11. In a JK flipflop, if J = 1 and K = 0, what happens to Q when the clocks occur?
   A. Q resets or stays reset
   B. Q sets or stays set
   C. Q toggles
   D. nothing (stays the same)

12. In a JK flipflop, if J = 1 and K = 1, what happens to Q when the clocks occur?
   A. nothing (stays the same)
   B. Q sets or stays set
   C. Q follows the polarity of the clock input
   D. Q toggles

13. A ring counter is:
   A. a shift register connected so data recirculates
   B. another name for a ripple counter
   C. several ORs connected output to input
   D. several XORs connected output to input

14. In an adjustable modulus counter:
   A. the propagation delay can be changed
   B. the divide-by-ratio can be changed
   C. the working voltage can be changed
   D.

15. A BCD counter resets to 0 and outputs a “carry” every ____ input pulses:
   A. 9
   B. 3
   C. 5
   D. 10

CHAPTER 5 – MICROCONTROLLERS AND PROGRAMMING

1. “MCU” means: (two correct answers)
   A. microcomputer unit
   B. microcirculator unit
   C. microcontroller unit
   D. machine control unit

2. In an MCU, calculations are done by:
   A. the instruction decoder
   B. the stack pointer
   C. the ALU
   D. the accumulator

3. Read-only memory:
   A. does not forget data when shut down
   B. usually can be written to only once
   C. is most of the memory in an MCU
   D. all of the above

4. An MCU communicates with the outside world via:
   A. the instruction bus
   B. the interrupt function
   C. RAM
   D. ports

5. A UART permits communication between:
   A. different parts of an MCU
   B. an MCU and devices using other data formats and timing
   C. RAM and ROM
   D. the ALU and the accumulator

6. An accumulator stores data currently being used by the:
   A. instruction decoder
   B. ALU
   C. watchdog timer
   D. stack

7. The program counter keeps track of:
   A. which instruction comes next
   B. how many instructions have been completed
   C. how many instructions are yet to be completed
   D. how many times a subroutine has been used

8. In an MCU, the program is usually stored in:
   A. ROM
   B. RAM
   C. the cache
   D. the control bus

9. An instruction cycle:
   A. is the time it takes to execute an instruction
   B. may take one clock cycle or several
   C. is counted by the watchdog timer
   D. A and B above

10. If precise timing is required, the clock generator will usually use:
   A. a 60Hz line frequency reference
   B. a radio tuned to WWV
   C. a crystal
   D. an LC tank

11. The MCU architecture where instructions and data use the same bus is:
   A. Van Der Rohe
   B. Harvard
   C. Von Neumann
   D. Stanford

12. In an EAPROM:
   A. data can be changed one byte at a time
   B. data can be erased by ultraviolet light
   C. data can be erased by a strong magnetic field
   D. data cannot be erased

13. In an EEPROM:
   A. data can only be changed by erasing everything and starting over
   B. data in a single register can be changed
   C. any bit in any register can be changed
   D. existing data is erased by ultraviolet light

14. Flash memory can be written to:
   A. dozens of times
   B. thousands of times
   C. unlimited number of times
   D. flash, like ROM, cannot be written to

15. A data direction register:
   A. controls whether port pins are set to send data or receive it
   B. controls whether data is flowing to or from the ALU
   C. controls whether a hard drive is reading or writing
   D. controls the routing of instructions

16. A flow chart is a diagram of:
   A. the current flows in a circuit
   B. the commands in a program
   C. the direction data travels on a bus
   D. how much current the MCU draws during various operations

17. A program loop is:
   A. a part of a program that repeats
   B. a cable with its ends plugged together
   C. a circular data path around a bus
   D. program instructions that tell part of the circuit to send data, then receive data

18. A subroutine is:
   A. an unimportant part of the program
   B. part of the program that is executed more slowly than the rest
   C. a small program called upon by the main program
   D. a program that isn't as good as most

19. Machine language:
   A. is binary code
   B. is binary code translated into words and abbreviations
   C. executes more slowly than a high level language
   D. is the easiest to learn and remember

20. Assembly language:
   A. is binary code
   B. is binary code translated into words and abbreviations
   C. requires a license to use
   D. is the same for all MCUs

21. A high level language:
   A. combines many machine-language commands in a single instruction
   B. is not tied to a particular MCU
   C. cannot be directly used by an MCU
   D. all of the above

22. A compiler:
    A. is the list of instructions an MCU can execute
    B. translates Assembly to a high level language
    C. translates a high level language to machine code
    D. keeps track of program bugs that need fixing

23. An interpreter:
    A. converts code between high level languages, e.g. C++ to Basic
    B. is a compiler that runs in real time
    C. is the fastest-running of all program types
    D. is the same for all MCUs

24. The part of an MCU that tells the system where to return to the main program after a subroutine is:
    A. the instruction decoder
    B. the last instruction in the subroutine
    C. the interrupt routine
    D. the stack

25. An interrupt tells the MCU to:
    A. stop and wait for further instructions
    B. stop what it's doing and go to a subroutine
    C. complete the regular program, then go to a subroutine
    D. skip certain program steps

26. When the ALU is performing operations that should not be interrupted, it can set a/an:
    A. barrier
    B. Interrupt Deny
    C. mask
    D. delay timer

CHAPTER 6 – COMPUTERS

1. A computer typically has the following, which an MCU doesn't have:
    A. changeable programs
    B. a user interface (keyboard, monitor etc)
    C. a very large amount of memory
    D. all of the above

2. “Main memory,” which accounts for most of the memory in a computer, consists of:
    A. ROM
    B. dynamic RAM
    C. static RAM
    D. read / write ROM

3. Most of the calculating and CMOS memory circuitry in a typical computer resides in/on:
    A. the hard drive
    B. a UART
    C. the mother board
    D. a peripheral

4. A daughter board is:
    A. a PC board connected to the mother board by a cable
    B. a PC board inside a peripheral
    C. found inside the power supply
    D. a small PC board that is mounted on the motherboard

5. An expansion slot is:
    A. a universal connector that can accept PC boards with any number of contacts
    B. a connector that provides for a second MPU
    C. a connector meant for a daughter board added by the user
    D. a box that adapts a single USB connector to multiple USB connectors

6. “MPU” means:
    A. microprocessor unit
    B. micropower unit
    C. microprocessor USB
    D. microprogramming unit

7. A hard disc drive:
    A. consists of spinning discs coated with ferromagnetic material
    B. records data as tiny magnetized areas on the coating
    C. often has 1,000x as much data storage as main memory
    D. all of the above

8. A piece of memory that stores a single binary word is called a/an:
    A. slot
    B. flash memory
    C. register
    D. file

9. Each location in memory has a/an_____ that the computer uses to keep track of and communicate with it.
    A. port
    B. address
    C. bus
    D. channel

10. A group of registers containing data that is to be used together as a set is called a/an:
    A. disc
    B. file
    C. USB drive
    D. subroutine

11. "OS" means:
    A. oversight system
    B. operating synthesis
    C. operating system
    D. originating system

12. dynamic RAM is so named because it:
    A. must be refreshed every few milliseconds
    B. draws more current than regular RAM
    C. runs faster than regular RAM
    D. only lasts for a few thousand read / write cycles

13. The cache is:
    A. memory that holds the most valuable data
    B. where return-from-subroutine addresses are stored
    C. memory which holds data that will soon be discarded
    D. memory built into the CPU

14. Retrieving data from memory is called a ____ operation.
    A. unload
    B. read
    C. query
    D. output

15. Storing data in memory is called a _____ operation.
    A. cache
    B. fill
    C. place
    D. write

16. Parity bits are used:
    A. to show that two numbers are equal
    B. to detect whether a communication channel is functional
    C. to detect and correct errors
    D. to configure the system for 8, 16, 32 or 64-bit words

17. A green 1/8" connector is a/an:
    A. microphone input
    B. audio line / headphone output
    C. line-level input
    D. subwoofer output

18. A 2-wire RS-232 data link is:
    A. half duplex
    B. full duplex
    C. simplex
    D. RS-232 requires more than 2 wires

19. A 2-way data link where the units on each end take turns sending is:
    A. stereo
    B. simplex
    C. full duplex
    D. half duplex

20. The maximum length of an RS-232 link is:
    A. 100 feet
    B. 1,000 feet
    C. depends on data rate
    D. depends on terminating impedance

21. The maximum length of a USB link is:
    A. about 15 feet
    B. about 5 feet
    C. about 50 feet
    D. about 500 feet

22. LVDS means:
    A. low voltage differential signaling
    B. low volume data storage
    C. linear voltage distribution system
    D. Las Vegas Department of Sanitation

23. LVDS, USB, Ethernet and CAN bus are alike in that they:
    A. are full duplex
    B. use balanced / differential wiring
    C. use more than 2 voltage levels
    D. require shielded cables

24. In a daisy chain:
    A. each node has its own cable back to the central computer
    B. each node's output is connected to the input of the next node in the chain
    C. there is always DC continuity throughout the entire chain
    D. there is always a complete loop through all nodes and back to the central computer

25. In a star configuration:
    A. all nodes must be plugged into the same AC outlet
    B. each node has its own cable connecting it to a central hub
    C. data voltages are +5V and -5V
    D. each node must have its own power supply

26. In a multidrop system:
    A. all nodes are connected in parallel to a single cable
    B. only one node's output can be in the “Z” state at a time
    C. nodes are never electrically isolated from the line
    D. nodes are never locally grounded

27. “Hot swapping” means:
    A. trading data between files
    B. switching among accumulators
    C. data transfer at rates above 1Gb/second
    D. connecting or disconnecting parts of the system while power is on

28. Another name for an 8P8C connector is:
    A. D connector
    B. Phoenix connector
    C. Coaxial connector
    D. RJ45 connector

29. In an Ethernet system, “10BASE-T” means:
    A. baseband, 10MBytes/second, twisted pair
    B. baseband, 10Mbits/second, twisted pair
    C. baseband, 10Gbits/second, twisted pair
    D. base frequency, 10Mbits/second, twisted pair

30. In 10BASE-T and 100BASE-TX, ___ cable pairs are used.
    A. 1
    B. 2
    C. 4
    D. 8

31. POE means:
    A. power-out emergency
    B. protocol of Ethernet
    C. power over Ethernet
    D. power-on enable

32. LAN means:
    A. local area network
    B. local access network
    C. large area network
    D. latency adjusting network

33. WAN means:
    A. work area network, e.g. an office cubicle
    B. Western area network, a digital protocol used in the Americas
    C. wide access network
    D. wide area net, e.g. a neighborhood

34. A packet is:
    A. another name for a file
    B. a chunk of data with routing and error correction information
    C. a portable device containing data, e.g. USB drive
    D. a protected file requiring an access code

35. In Ethernet, packets are called:
    A. bundles
    B. Bytes
    C. addresses
    D. frames

36. In Thunderbolt / Display port, there are ___ parallel channels called _____.
    A. 2, routes
    B. 4, channels
    C. 4, lanes
    D. 6, lanes

37. In LVDS, the data link takes the form of:
    A. + and – 5V levels
    B. a 3.5mA current loop
    C. short 1V pulses
    D. a fiber optic cable

38. Two signals or data streams operating “in quadrature” are:
    A. 180 degrees out of phase
    B. 90 degrees out of phase
    C. opposite polarity
    D. opposite direction

CHAPTER 7 – MIXED ANALOG / DIGITAL

1. “Digitization” means:
    A. converting base-10 values to base-2 values
    B. burning a CD-ROM
    C. converting digital values to analog
    D. converting analog values to digital

2. “Sampling” means:
    A. measuring an analog voltage at a particular moment
    B. measuring an analog voltage over a long period of time
    C. measuring the frequency content of a signal
    D. measuring the noise present in a signal

3. “Sampling rate” means:
   A. how quickly an analog sample is converted to digital
   B. how many samples are taken per second
   C. the ratio of the sampling frequency to the highest signal frequency
   D. how accurate each sample is

4. “Bit depth” means:
   A. the voltage represented by one bit
   B. how much error the MSB can have
   C. how many samples are taken each second
   D. how accurate each sample is

5. “Quantization error” means
   A. the difference between when a sample is taken and when it's supposed to be taken (i.e. jitter)
   B. the difference between a quantized value and the analog voltage's actual value
   C. the error introduced when data is compressed
   D. the DC component of a signal that's supposed to be AC only

6. An A-to-D converter:
   A. converts digital values to analog
   B. converts linear quantizing to nonlinear
   C. converts analog values to digital
   D. converts 1-bit delta samples to 16- or 24-bit words

7. The D-to-A converter:
   A. converts digital values to analog
   B. converts nonlinear quantizing to linear
   C. converts analog values to digital
   D. converts 1-bit delta samples to 16- or 24-bit words

8. A digital voltmeter typically uses a/an _____ A-D.
   A. successive approximation
   B. integrating
   C. flash
   D. delta

9. “SAR” means”
   A. sequential approximation register
   B. successive assortment register
   C. successive approximation register
   D. successive arithmetic register

10. In a SAR, each approximation reduces quantizing error by:
   A. 12dB
   B. 10%
   C. 1/4
   D. 1/2

11. A flash A-D is based on a large number of:
   A. AND gates
   B. OR gates
   C. integrators
   D. comparators

12. The output of a delta A-D represents:
   A. the voltage change from one sample to the next
   B. the cumulative voltage change over time
   C. change in bit depth between one sample and the next
   D. change in sample rate between one sample and the next

13. To properly sample and reconstruct a 10kHz sine wave, we would need _____ samples per second.
   A. 40,000
   B. 20,000
   C. 10,000
   D. 5,000

14. In linear quantizing:
   A. any existing distortion in the analog waveform is removed
   B. quantizing steps are always the same size
   C. the data stream consists of 1-bit samples
   D. the Nyquist limitation does not apply

15. In nonlinear quantizing
   A. the sample rate varies
   B. serious waveform distortion is introduced
   C. small voltages are typically quantized more finely than large voltages
   D. large voltages are typically quantized more finely than small voltages

16. Two traditional nonlinear quantizing standards are: (two correct answers)
   A. natural law
   B. maritime law
   C. A-law
   D. μ-law

17. “DSP” means:
   A. digital signal proportioning
   B. digital signal processing
   C. digital sample processing
   D. delta signal processing

18. True or false: DSP can perform all the processing functions we can perform with analog circuitry, and more
   A. true
   B. false

19. A comb filter:
    A. removes energy at a number of frequencies from a signal
    B. removes energy at a single frequency from several signals at once
    C. removes low frequencies
    D. removes high frequencies

20. "Latency" means:
    A. fundamental nonlinearity of an A-D
    B. energy remaining after digitization, resulting in echo
    C. delay between signal input and output
    D. residual magnetism

21. "Data compression" means:
    A. storing the same information in fewer Bytes
    B. removing every other Byte
    C. increasing the clock speed so each Byte takes less time
    D. reducing bit depth

22. "Lossless compression" means:
    A. latency is zero
    B. the bit error rate is zero
    C. data can be re-quantized in a different format
    D. all the original data can be restored

23. "Lossy compression" means:
    A. the bit rate is slowed down
    B. error correction is not used
    C. some data is lost and cannot be restored
    D. each samples is truncated, i.e. least significant bits are discarded

24. "Chroma subsampling" means:
    A. hiding color information as a hidden code within brightness information
    B. sampling color with less detail than brightness
    C. sampling color with smaller bit depth than brightness
    D. transmitting color on a lower carrier frequency than brightness

CHAPTER 8 – MACINE CONTROL

1. "CNC" means:
    A. compiled numeric control
    B. computer numeric control
    C. computed numeric control
    D. computer natural code

2. In an open loop system, data flows:
    A. from machine to controller
    B. wirelessly
    C. from controller to machine
    D. from controller to ground

3. In a closed loop system, data flows:
    A. to all machines in parallel
    B. from controller to machine and machine to controller
    C. from sensor directly to motor
    D. to one machine after another, i.e. daisy chain

4. Binary 101001 is _____ in Gray code.
    A. 110111
    B. 011011
    C. 101101
    D. 111101

5. A hot chassis situation occurs when:
    A. the frame of a machine is connected to the AC power source
    B. the frame of a machine is connected to a controller's +5V DC power source
    C. the controller is forced to carry more current than it can handle
    D. lightning strikes

6. "PLC" means:
    A. programmed logic controller
    B. progressive logic computer
    C. programmable ladder controller
    D. programmable logic controller

7. "Hardened" refers to a controller that:
    A. has a steel case
    B. can work in a hostile environment
    C. cannot be accidentally reprogrammed
    D. retains its program when power is turned off

8. At a minimum, connections to a PLC consist of:
    A. Y inputs, X outputs, power, programming
    B. X inputs, Y outputs, power, programming
    C. USB port, Ethernet port, power
    D. relay inputs, optically isolated outputs, power

9. A DIN rail:
    A. is a U-shaped channel
    B. is designed to hold PLC and related units
    C. can be cut to length with a hacksaw
    D. all of the above

10. Ladder logic is a graphic way to:
    A. show analog to digital conversion
    B. show power consumption at various points in the system
    C. develop a machine control program
    D. show system wiring; i.e. a schematic diagram

11. Using a normally open pushbutton with a controller to stop a machine is poor practice because:
    A. it may not be able to handle the current
    B. if an open occurs in the circuit, you can't turn the machine off.
    C. you can't get a N.O. pushbutton in "stop" red
    D. normally open pushbuttons cost more

12. A relay:
    A. uses a small current to switch a large current
    B. provides electrical isolation
    C. can switch any AC or DC voltage and current (within limits)
    D. all of the above

13. A zero crossing switch turns on and off
    A. during the AC waveform's positive peaks
    B. during the AC waveform's negative peaks
    C. during the AC waveform's 0-volt points
    D. during the AC waveform's 0-current points

14. Optical isolation between controller and a large MOSFET or IGBT may be used to:
    A. protect the controller
    B. change levels
    C. avoid a hot chassis
    D. all of the above

15. A transistor's safe operating area defines:
    A. where a transistor may be installed in a circuit
    B. the range of voltage and current that will not damage the transistor
    C. the highest frequency that will not damage the transistor
    D. the heat sink area necessary to keep the transistor cool

16. In a PWM motor control circuit, duty cycle is varied to:
    A. vary the output voltage
    B. vary the output frequency
    C. vary the 3rd harmonic content
    D. correct for waveform distortion

17. "VFD" means:
    A. variable frequency device
    B. verifiable frequency drive
    C. variable frequency drive
    D. voltage-forward drive

18. A VFD is used to control the speed of:
    A. a DC motor
    B. an AC motor
    C. an AC generator
    D. a DC generator

19. "MUX" means:
    A. multiplexing
    B. maximizing
    C. multi-crossing
    D. multi-mixing

20. A MUX permits a single analog input to:
    A. distribute a signal to many places
    B. be translated to many forms, e.g. USB, Ethernet, RS485 etc
    C. measure a signal that has no ground reference
    D. monitor voltages at many points

21. Using high-current switch contacts in a low-current (dry) sensing circuit is a bad idea because:
    A. the switch may generate a small voltage of its own
    B. high-current contacts are needlessly expensive
    C. they need substantial current to stay clean
    D. they will wear faster than low-current contacts

22. The main virtue of the Hall effect switch is:
    A. it is not affected by dirt or grease
    B. it is inexpensive
    C. it can sense from a great distance
    D. it is more accurate than any other switch

23. A desirable characteristic of a photoresistive cell is:
    A. it is the least expensive sensor
    B. it is electrically shielded
    C. it can conduct AC, or DC of either polarity
    D. it is made of non-toxic materials

24. An undesirable characteristic of a photocell is:
    A. it is comparatively slow
    B. it requires a high voltage to operate
    C. it can leak oil
    D. it requires complicated support circuitry

25. In an AC circuit, flyback can be tamed by connecting a series RC string:
    A. between common and Earth ground
    B. between the “hot” wire and Earth ground
    C. between the “hot” wire and chassis
    D. across the switch contacts

26. Moving a certain distance from a defined starting position is called:
    A. poor registration
    B. indexing
    C. vibration
    D. convolution

27. A device that puts out a certain number of pulses with each rotation is called a/an:
    A. signal generator
    B. potentiometer
    C. stepper motor
    D. tachometer

28. A _____ motor rotates a specific number of degrees per electrical pulse
    A. DC
    B. stepper
    C. AC
    D. linear

29. A binary-like code that changes by only one bit between values is called:
    A. Cyclic code
    B. Gray code
    C. Hammurabi code
    D. A and B above

30. A moving system which greatly overshoots its target is:
    A. overpowered
    B. overlubricated
    C. underpowered
    D. underdamped.

31. “PID” means:
    A. pulse integral drive
    B. proportional, instantaneous, differential
    C. proportional, integral, differential (or derivative)
    D. power / current drive

32. When turn-on and turn-off points are different, a machine has:
    A. error
    B. loose sensors
    C. wobble
    D. hysteresis

33. A large spark when a switch opens is usually due to:
    A. high voltage
    B. inductive flyback
    C. a short circuit
    D. a bad switch

CHAPTER 9 – DISPLAYS AND PRINTERS

1. A 2-dimensional image is sent from one place to another by:
    A. holography
    B. converting to a 1-dimensional stream
    C. modulating on two wires
    D. converting a data stream to sine waves

2. A traditional (analog) video picture is built up of 1-dimensional, almost horizontal lines. This is called a/an:
    A. file
    B. raster
    C. interlace
    D. phosphor

3. Sending signals to pixels in sequence, one after another, is called:
    A. time division multiplexing
    B. time domain multiplying
    C. television display matrixing
    D. time derived multiplexing

4. Red, green and blue are known as:
    A. solid colors
    B. independent colors
    C. primary colors
    D. saturated colors

5. A video transmission standard that combines brightness, color and synchronization signals into a single analog signal is called:
    A. broadcast video
    B. public video
    C. composite video
    D. integrated video

6. S-video sends ___ and ___ over separate wires.
    A. luminance, chrominance
    B. luminance, synchronization
    C. picture, synchronization
    D. horizontal sync, vertical sync

7. In a digital display, the number of horizontal and vertical pixels defines the: (two correct answers)
   A. correlation
   B. definition
   C. frame rate
   D. resolution

8. The ratio of a display's width vs. height is called:
   A. the size ratio
   B. the aspect ratio
   C. portrait or landscape format
   D. the panorama

9. In component video, the signals carried on separate shielded wires are:
   A. red, green,blue, luminance, carrier
   B. red, green, blue, luminance, chrominance
   C. red, green, blue, horizontal sync, vertical sync
   D. luminance, chrominance, horizontal sync, vertical sync, ground

10. DVI uses ___ balanced pairs for video.
   A. 2
   B. 3
   C. 6
   D. 12

11. Standard HDMI uses ___ balanced pairs for video.
   A. 3
   B. 4
   C. 6
   D. 9

12. True or false: Displayport's lanes do not necessarily correspond to red, green and blue.
   A. true
   B. false

13. LCD display panels use _____ filters.
   A. decimating
   B. dichroic
   C. polarizing
   D. comb

14. An LCD controls light transmission by:
   A. twisting polarized light
   B. blocking light
   C. diverting light sideways
   D. reflecting light back to the source

15. Most LCD panels do not have a wide viewing angle because:
   A. the dichroic filters focus the light forward
   B. the lenticular lens focuses light forward
   C. polarized light is invisible at an angle
   D. the polarizing filters focus light forward

16. A plasma display works by:
   A. using electricity to excite red, green and blue gasses
   B. turning on millions of organic LEDs
   C. using fluorescent light to excite phosphor-coated pixels
   D. turning on parts of an electroluminescent panel

17. In a 3-LCD or DLP projector, white light is split into red, green and blue using:
   A. comb filters
   B. dichroic mirrors
   C. lenticular lenses
   D. compound lenses

18. In a single-LCD or DLP projector, colors are obtained by:
   A. fixed red, green and blue filters
   B. dichroic mirrors
   C. lenticular lenses
   D. a color wheel

19. In an electroluminescent panel, the phosphor is lit up by:
   A. the field created by an AC voltage
   B. the heat created by an AC current
   C. electrostatic attraction
   D. electromagnetic attraction

20. “Electronic paper” (e.g. Nook, Kindle etc) is based on:
   A. chemical reactions
   B. miniature gears
   C. moving particles
   D. glowing particles

21. A thermal printer creates images using:
   A. a laser
   B. special heat-sensitive paper
   C. heated ink droplets
   D. magnetism

22. Two types of inkjet printer heads are: (two correct answers)
   A. magnetic
   B. thermal
   C. piezoelectric
   D. pressurized air

23. A continuous-jet printer uses ____ to control the stream of ink
   A. electromagnetic deflection
   B. electrostatic deflection
   C. a moving vane
   D. air pressure

24. Xerography and laser printers are based on:
    A. magnetic ink
    B. capacitive coupling
    C. static electricity and photoconductivity
    D. particle physics

25. In a color printer, the ink colors are:
    A. red, green, blue, black
    B. red, yellow, blue, black
    C. orange, green, violet, black
    D. cyan, magenta, yellow, black

26. Black is called “K” because
    A. it's the image that the colors key to
    B. otherwise it might be confused with blue
    C. the ink was first developed by Kodak
    D. the ink's chemical base is ketone

CHAPTER 10 – SERVICING DIGITAL EQUIPMENT

1. Cable attenuation is worse with: (two correct answers)
    A. unshielded wires
    B. longer cables
    C. higher signal frequencies
    D. twisted pairs

2. Background molecular noise is most likely to cause problems with:
    A. high-speed systems
    B. balanced cables
    C. shielded wires
    D. wireless systems

3. High-frequency cable attenuation:
    A. causes smearing of 0-1 and 1-0 transitions
    B. places a limit on the highest bit rate
    C. places a limit on cable length
    D. all of the above

4. An improper cable termination causes:
    A. skew
    B. reflections
    C. jitter
    D. latency

5. Ringing is most often caused by:
    A. an impedance mismatch
    B. too high a frequency in a data bus or cable
    C. using the wrong family of logic ICs
    D. a circuit resonance at some frequency

6. A momentary change in a ground's potential is called:
    A. ground bump
    B. a ground loop
    C. bypassing
    D. an open ground

7. A momentary change in the +DC supply voltage is called:
    A. intermodulation
    B. distortion
    C. overload
    D. rail bump

8. A timing discrepancy between two data streams that are supposed to be synchronized is called:
    A. jitter
    B. skew
    C. latency
    D. quantizing error

9. Randomly varying timing discrepancies are called:
    A. quantizing error
    B. jitter
    C. skew
    D. latency

10. When working on the insides of a piece of equipment, ESD damage is avoided by:
    A. using an isolation transformer in the AC supply
    B. wearing rubber gloves
    C. grounding bench top and technician (through 1M resistors)
    D. turning off all nearby equipment

11. A logic pulser:
    A. injects a 1 or 0 into a circuit
    B. injects +5V or -5V into a circuit
    C. injects an ASCII sequence
    D. acts as a valve

12. A logic probe:
    A. shows activity at a point in the circuit
    B. shows ripple on the +DC supply
    C. measures cable impedance
    D. detects bit errors in a data stream

13. A current probe:
    A. measures current in the +DC rail
    B. measures how much current each IC is drawing
    C. detects the magnetic field created by data flowing in a wire or PC trace
    D. detects bit errors in a data stream

14. Active probes can be ruined by:
    A. overheating
    B. connecting to a voltage higher than 15V
    C. placing it in a strong magnetic field
    D. connecting two probes together

15. A logic analyzer display shows:
    A. sine waves at different frequencies
    B. a Boolean gate diagram
    C. Gray code
    D. a number of data streams

16. A spectrum analyzer shows a graph of:
    A. voltage vs. time
    B. voltage vs. frequency
    C. voltage vs. phase
    D. voltage vs. impedance

17. The unmodulated signal of a radio or TV station is called a/an:
    A. baseband
    B. harmonic
    C. carrier
    D. synchronization

18. Modulating a carrier produces
    A. sidebands
    B. distortion
    C. carrier frequency drift
    D. interference

19. Harmonic analysis shows:
    A. the waveshape of a signal
    B. the frequency of a signal
    C. the harmonic content of a signal
    D. whether a signal matches a reference signal

20. Intermodulation analysis shows:
    A. the harmonic content of two signals
    B. new frequencies created by two signals
    C. short-duration voltage peaks
    D. how electromagnetic fields interfere with a signal

21. A limitation of the sweep method of spectrum analysis is:
    A. it cannot display a wide frequency spectrum
    B. it takes a very long time
    C. it will miss short-term events
    D. it cannot be used with balanced lines

22. A limitation of the Fourier transform method of spectrum analysis is:
    A. signals that do not correspond to a bin frequency may seem to be attenuated
    B. low frequencies cannot be displayed
    C. any 60Hz hum screws up the measurement
    D. it's extremely expensive

23. “Sampling noise gain” makes the noise seem:
    A. lower than it actually is
    B. higher than it actually is
    C. to be concentrated near one frequency
    D. to rise at 6dB per octave

24. Oscilloscope waveform “crawl” is most easily avoided by:
    A. Earth-grounding the scope
    B. synchronizing the scope to the signal
    C. setting a very fast horizontal sweep
    D. carefully tweaking the vertical gain control

25. The control used to avoid crawl is called the:
    A. input AC / ground / DC switch
    B. horizontal range switch
    C. trigger
    D. vertical range switch

26. If a scope trace's vertical dimension is too small or large to see clearly, the user adjusts:
    A. the vertical range switch
    B. the horizontal range switch
    C. the trigger
    D. the focus control

27. If a 10V DC voltage has 10mV of ripple on it, the ripple can be closely examined by:
    A. setting the probe attenuator to 10X
    B. setting the input switch to AC
    C. setting a slow sweep speed
    D. Earth-grounding he scope

28. In digital servicing, the main benefit of a dual-trace oscilloscope is that:
    A. if one channel breaks, you still have the other channel to look at waveforms
    B. it permits timing comparison of two data streams
    C. it can display Lissajous patterns
    D. it is good at displaying sine waves

29. Two things a digital scope can do that most analog scopes cannot do are: (two correct answers)
    A. capture and store a long data stream
    B. display in real time
    C. perform spectrum analysis
    D. Fourier analysis

30. Two things an analog scope can do that lower-priced digital scopes cannot do are: (two correct answers)

A. display in real time
B. Fourier analysis
C. memorize a data stream
D. display very high frequencies

31. A vector analyzer shows:

A. amplitude and impedance
B. amplitude and frequency
C. voltage vs. frequency
D. amplitude and phase

32. A data analyzer does *not* show:

A. voltage vs. frequency
B. voltage vs. time
C. phase
D. impedance

33. Two methods of digital signal generation are: (two correct answers)

A. real-time calculation
B. scanning a lookup table
C. diode wave shaping
D. resonant oscillation

34. A place in a circuit that the manufacturer recommends as a place to look at a data stream is called a/an:

A. DC power supply plug
B. loading point
C. test point
D. constellation point

35. A diagram that shows functional sections of a circuit is called a/an”

A. manufacturer's parts diagram
B. block diagram
C. schematic diagram
D. Venn diagram

36. A diagram that shows actual parts and connections is called a/an:

A. pinout diagram
B. block diagram
C. schematic diagram
D. logic diagram

37. Power over Ethernet can cause problems because:

A. the main power supply can be overloaded
B. different manufacturers use different connections
C. supply current and data can interfere with each other
D. expensive special cables are required

38. If cable attenuation is too great for the system to work reliably, one cure is to install a/an:

A. alternator
B. terminator
C. repeater
D. parity checker

ANSWERS

Chapter 1, Digital Concepts

1-B 2-A 3-B 4-D 5-D 6-D 7-C 8-C 9-B 10-D 11-D 12-B 13-A 14-D 15-A 16-B 17-B 18-A 19-D 20-B 21-A 22-A 23-C

Chapter 2, Binary Arithmetic

1-A,D 2-B 3-C 4-C 5-B 6-C 7-B 8-B 9-A,C 10-A 11-B 12-B 13-C 14-D 15-A 16-C 17-B 18-A 19-A 20-B 21-B

Chapter 3, Boolean Logic

1-B 2-B 3-C 4-B 5-C 6-D 7-B 8-A 9-D 10-A 11-B 12-B 13-C 14-A 15-C 16-B 17-B 18-D 19-B 20-A 21-D 22-B 23-B 24-C 25-C

Chapter 4, Sequential Logic

1-B 2-A 3-C 4-C 5-D 6-B 7-D 8-B,C 9-A 10-A 11-B 12-D 13-A 14-B 15-D

Chapter 5, Microcontrollers and Programming

1-A,C 2-C 3-D 4-D 5-B 6-B 7-A 8-A 9-D 10-C 11-C 12-A 13-A 14-B 15-A 16-B 17-A 18-C 19-A 20-B 21-D 22-C 23-B 24-D 25-B 26-C

Chapter 6, Computers

1-D 2-B 3-C 4-D 5-C 6-A 7-D 8-C 9-B 10-B 11-C 12-A 13-D 14-B 15-D 16-C 17-B 18-C 19-D 20-C 21-A 22-A 23-B 24-B 25-B 26-A 27-D 28-D 29-B 30-B 31-C 32-A 33-D 34-B 35-D 36-C 37-B 38-B

Chapter 7, Mixed Digital-Analog

1-D 2-A 3-B 4-D 5-B 6-C 7-A 8-B 9-C 10-D 11-D 12-A 13-B 14-B 15-C 16-C,D 17-B 18-A 19-A 20-C 21-A 22-D 23-C 24-B

Chapter 8, Machine Control

1-B 2-C 3-B 4-D 5-A 6-D 7-B 8-B 9-D 10-C 11-B 12-C 13-C 14-D 15-B 16-A 17-C 18-B 19-A 20-D 21-C 22-A 23-C 24-A 25-D 26-B 27-C 28-B 29-D 30-D 31-C 32-D 33-B

Chapter 9, Displays and Printers

1-B 2-B 3-A 4-C 5-C 6-A 7-B,D 8-B 9-C 10-C 11-A 12-A 13-C 14-A 15-B 16-C 17-B 18-D 19-A 20-C 21-B 22-B,C 23-B 24-C 25-D 26-A

Chapter 10, Servicing Digital Equipment

1-B,C 2-D 3-D 4-B 5-D 6-A 7-D 8-B 9-B 10-C 11-A 12-A 13-C 14-B 15-D 16-D 17-C 18-A 19-C 20-B 21-C 22-A 23-A 24-B 25-C 26-A 27-B 28-B 29-A,C 30-A,D 31-D 32-D 33-A,B 34-C 35-B 36-C 37-B 38-C

End of EM4 Practice Exam